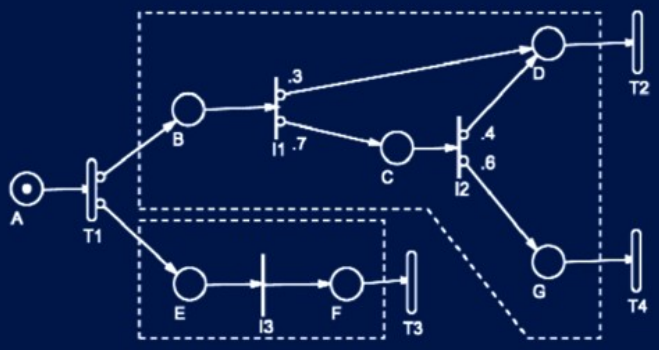
Tutorial

LNCS 2090

Ed Brinksma
Holger Hermanns
Joost-Pieter Katoen (Eds.)

Lectures on Formal Methods and Performance Analysis

First EEF/Euro Summer School on Trends in Computer Science



Springer

Lecture Notes in Computer Science 2090

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Ed Brinksma Holger Hermanns
Joost-Pieter Katoen (Eds.)

Lectures on Formal Methods and Performance Analysis

First EEF/Euro Summer School
on Trends in Computer Science
Berg en Dal, The Netherlands, July 3-7, 2000
Revised Lectures

Springer

Volume Editors

Ed Brinksma
Holger Hermanns
Joost-Pieter Katoen
University of Twente, Faculty of Computer Science, Formal Methods and Tools Group
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: {brinksma/hermanns/katoen}@cs.utwente.nl

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Lectures on formal methods and performance analysis : revised lectures /
First EEF Summer School on Trends in Computer Science, Berg en Dal, The
Netherlands, July 3 - 7, 2000. Ed Brinksma ... (ed.). - Berlin ; Heidelberg ;
New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ;
Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2090)
ISBN 3-540-42479-2

CR Subject Classification (1998): F.3, D.2, C.4, C.2.4, D.3

ISSN 0302-9743
ISBN 3-540-42479-2 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 10839370 06/3142 5 4 3 2 1 0

Foreword

Traditionally, models and methods for the analysis of the functional correctness of reactive systems, and those for the analysis of their performance (and dependability) aspects, have been studied by different research communities. This has resulted in the development of successful, but distinct and largely unrelated modeling and analysis techniques for both domains. In many modern systems, however, the difference between their functional features and their performance properties has become blurred, as relevant functionalities become inextricably linked to performance aspects, e.g. isochronous data transfer for live video transmission.

During the last decade, this trend has motivated an increased interest in combining insights and results from the field of *formal methods* – traditionally focused on functionality – with techniques for *performance modeling and analysis*. Prominent examples of this cross-fertilization are extensions of process algebra and Petri nets that allow for the automatic generation of performance models, the use of formal proof techniques to assess the correctness of randomized algorithms, and extensions of model checking techniques to analyze performance requirements automatically. We believe that these developments mark the beginning of a new paradigm for the modeling and analysis of systems in which qualitative and quantitative aspects are studied from an integrated perspective. We are convinced that the further work towards the realization of this goal will be a growing source of inspiration and progress for both communities.

The aim of the EEF summerschool on *Formal Methods and Performance Analysis* (FMPA) was to report on the state-of-the-art research and tool development for the integrated modeling and analysis of functional and performance aspects of reactive systems. To provide the necessary background it also included lectures on basic models and techniques of both performance evaluation and formal methods for reactive systems. The lectures were given by internationally recognized experts from the formal methods and performance analysis communities. These invited lecturers were: Christel Baier (*Model checking probabilistic and Markovian models*), Gianfranco Balbo (*Petri nets and stochastic Petri nets*), Ed Brinksma (*Process algebra*), Christos Cassandras (*Discrete event simulation*), Gianfranco Ciardo (*Structured and distributed analysis*), Reinhard German (*Non-Markovian analysis*), Boudewijn Haverkort (*Markov chain models and analysis*), Holger Hermanns (*Markovian process algebra*), Ulrich Herzog (*Formal methods for performance analysis*), Jane Hillston (*Compositional and decompositional analysis*), Joost-Pieter Katoen (*Non-Markovian process algebra*), William Sanders (*Stochastic activity networks and their analysis*), Roberto Segala (*Verification of probabilistic distributed algorithms*) and Pierre Wolper (*Model checking*).

This LNCS volume contains a series of articles by lecturers at the summerschool, which survey most of the topics covered at the school, as well as some

additional, related material. We believe that this volume will be of considerable interest to researchers from both the formal methods and performance analysis communities, and that it should prove an excellent starting point for those who wish to get acquainted with the research at the crossroads of these fields.

FMPA was organized as the first school on Trends in Computer Science by the European Educational Forum (established in 1996), a European research training initiative focusing on basic research in Computer Science and its applications. EEF has partner organizations from 7 countries (Denmark, The Netherlands, Finland, United Kingdom, Italy, Germany, France) which together involve 34 universities. The primary aim of the EEF is the training of Ph.D. students and young researchers. The training activities include workshops, schools, highly focused conferences, as well as conferences that provide a forum for a variety of topics of current interest. For more information, see the EEF web page: `http://www.tucs.abo.fi/EEF/`.

FMPA was held at Hotel Val Monte in Berg en Dal, a beautiful village close to Nijmegen. The school was very well attended with 80 participants from all over the world, with 36 attendees who were sponsored through the High-Level Scientific Conference Programme of the European Commission. Other sponsors were the Dutch National Graduate School IPA (Institute for Programming research and Algorithmics), the Netherlands Organization for Scientific Research (NWO), the Royal Dutch Academy of Sciences (KNAW), and the Center for Tele-Informatics and Information Technology (CTIT).

We would like to thank all lecturers for their excellent lectures and their high-level contributions to these lecture notes. We thank Jos Baeten, Tijn Borghuis, and Grzegorz Rozenberg for inviting us to organize FMPA as part of the EEF summerschool series and for their assistance in maintaining the proper contacts with the EC and Springer-Verlag. On the local level, we thank Pedro D'Argenio and in particular Joke Lammerink for their assistance with the organization of the school.

May 2001 Ed Brinksma, Holger Hermanns, Joost-Pieter Katoen

Table of Contents

Formal Methods for Performance Evaluation

Ulrich Herzog

Universität Erlangen-Nürnberg, Institut für Informatik 7, Rechnernetze und Kommunikationssysteme, Martensstr. 3,
D-91058 Erlangen, Germany
herzog@informatik.uni-erlangen.de

Abstract. The main goal of this contribution is to advocate the increased use of formal methods (FM) in the field of performance evaluation (PE). Moreover, we try to reduce the mutual reservations between both areas, formal specification techniques and performance evaluation since both can profit from such an integration: FMs may find their way into a new and very attractive area of applications and some fundamental problems of PE may be overcome.
The first part summarizes the evolution of PE, its methodology and the basic concepts of performance modeling and analysis, elaborated in specific contributions of this book.
Classical modeling and analysis techniques have a high standard and have been quite successful. However, there are important problem classes still open, and there are some fundamental deficiencies: Task interdependencies and synchronization, interfacing in modeling hierarchies, methods and tools for automating the performance engineering process are typical examples.
We therefore advocate the integration of FMs and PE and survey three advanced approaches, again, treated in detail in specific contributions: Stochastic Petri-Nets, Stochastic Activity Networks and Stochastic Process Algebras.
We try to summarize our own experience with these techniques and conclude with a list of challenging topics and current research directions.

1 Introduction

Performance Evaluation (PE) means to investigate and optimize the dynamic time-varying behavior within and between the individual components of transportation and processing systems. This includes the measurement and modeling of real system behavior, the definition and determination of characteristic performance measures, and the development of design rules which guarantee an adequate quality of service.

PE has a long tradition when designing, dimensioning and operating telecommunication systems. Traffic engineering teams are vital for all large companies in telecom-industry: They ensure the economic use of transmission and switching facilities, they are needed to assure the desired quality of service. Table 1 shows in the upper part some pioneers of this field (however, this list is incomplete and should contain a hundred or more names). During World War II, a

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 1–37, 2001.

second field of activities, operations research, emerged: The allocation of scarce resources to the various military operations was extremely important and teams of researchers were appointed to investigate these problems, many of them related to PE - cf. middle part of table 1. The results were greatly appreciated and are today standard in business management and production. In the sixties, people from industry and universities, cf. also table 1 bottom, tried to establish PE techniques in the world of computers and computer communication systems. Many fundamental results and methods were found. Unfortunately, however, they are not common knowledge: Computer scientists are completely focused on the functional behavior of computers, system software and application programs, the "insularity of PE" [17] is still the normal situation. "Computer scientists do not have a feeling for time" is a common saying getting to the heart of the problem. And we have to work hard to change this.

Table 1. Some Pioneers in PE.

Erlang	1908/18	telephone traffic	fundamental delay- and loss formulas
Palm	1943	telephone traffic	long-term variations
Jacobaeus	1950	switching networks	congestion in link systems
Clos	1953	switching networks	nonblocking system
Wilkinson	1955	toll traffic engineering	alternate routing systems
Cobham	1954	operations research	priority assignment
Jackson	1957	operations research	queuing in networks
Conway	1958/67	operations research	scheduling
Scherr	1965	time-sharing systems	measurement and modeling
Kleinrock	1964/74	ARPA (-> Internet)	performance and reliability
Buzen	1971	computers	central server model
Bux	1981	token ring network	performance simulation
Bellcore	mid 80ies	internet traffic	long-range dependency

There are many success stories of PE documented or informally reported coming from all three areas mentioned above [67,42,35,36]:

1. The economic dimensioning of national and international telecommunication networks,
2. the control and optimization of the Polaris missile development program, or
3. the efficient design of processor architectures and memory hierarchies in both industry and academia.

However, since systems consist of hardware and software, it is also common to fully design, implement and functionally test them before an attempt is made to determine their performance characteristics. The redesign of both software and hardware is costly and may cause late system delivery. Dramatic examples with enormous financial consequences are known. Figure 1 illustrates the impact of such design faults on the system life cycle: Steadily, the functional properties will

be detailed and added to the design until the implementation is operational (solid curve). Often, however, the first test shows that - although functionality has been checked - the overall behavior is completely unsatisfactory and a major redesign is necessary; the figure also shows a second loop back until the ideal line (dotted) is approached. Main cause is obviously and mostly that the system dynamics were neglected: Temporal analysis and assurance were performed but not until things had gone wrong! Figure 1 makes clear that the early and systematic integration of PE into the design process should be obligatory.

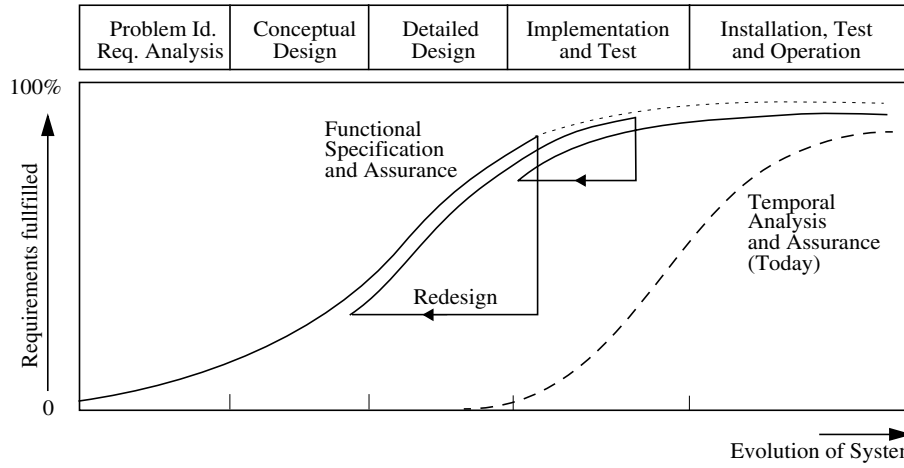


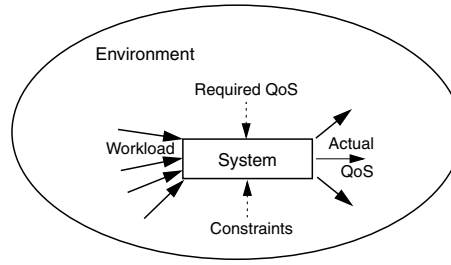
Fig. 1. System Life Cycle and Quality Assurance.

The first Euro Summer School On Trends In Computer Science was dedicated to this goal. This contribution tries to survey the basic concepts and techniques leading to the various contributions of this volume.

2 Methodological Steps

As we have seen in section 1, the objectives of PE are the modeling, the analysis and the synthesis of optimal system structures and dynamics. We measure and model the temporal behavior of real systems, define and determine characteristic performance measures, and develop design rules, which guarantee an adequate quality of service. A general scenario is shown in Figure 2: The environment generates requests, the so called workload, to the system:

1. The workload represents the sum of all needed and desired activities and services.
2. The system consists of one or more components trying to satisfy these requests.



Workload:
Sum of all needed or desired activities and services
- Type of activities
- Sequential ordering
- Time of requests
- Frequency, ...

Quality of Service (QoS):
Fitness for use. Achieved correspondence with user-requirements concerning
- Qualitative parameters: logical sequencing, liveness, priorities, ...
- Quantitative parameters: throughput, response time, ...

Constraints: Economic, functional, HW/SW-Technology, geographical, morphological, ...

Fig. 2. The System with its Environment, Requirements and Constraints.

3. An optimal system structure and operating mode is reached if the system fulfills all requirements concerning QoS as well as all technical and economic constraints.

To assure an appropriate performance today's PE-methodology includes the following steps, cf. Figure 3: Workload characterization and system parameter specification are the first sensitive steps. Determining these values needs care and knowledge about both the application and the technical system components. Next, the design methodology distinguishes between two totally different but complementary approaches: experiments on the real system (measurements) and modeling.

Both are followed by analysis steps using methods of statistics, stochastic processes and simulation. The validation of experimental and modeling results follows next and is very important [1]. Finally, system structures and operating modes are synthesized; systematic parameter variation (in case of experimentation and simulation) and mathematical optimization techniques (in combination with stochastic models) guarantee good or even optimal system design considering costs and/or performance and a variety of optimization constraints.

[1] We crosscheck measurements by modeling results and vice versa. Plausibility considerations and the study of limits are also very helpful

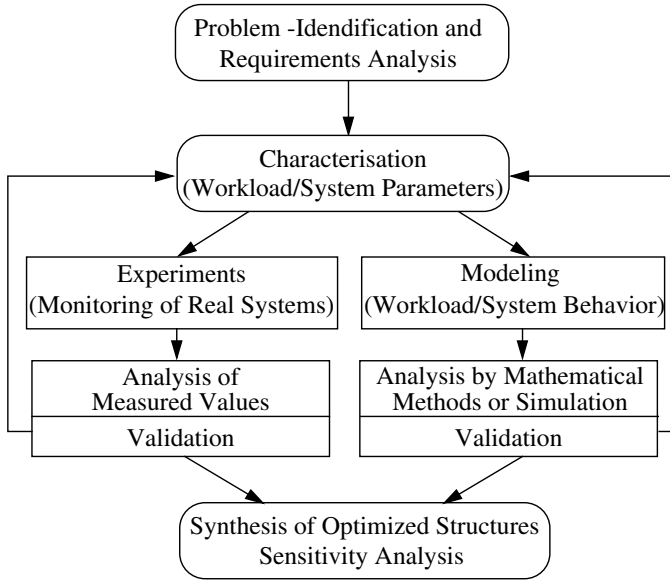


Fig. 3. Overview of Performance Evaluation Methodology.

By following all steps of the PE-methodology, considerable success in improving real system behavior, is possible. Some highlights from our own experience are

1. We fully generated a TCP/IP protocol stack from a high-level specification written in SDL. This resulted in a extremely slow communication between two SUN-WS. Carefully performed measurements in several refinement steps showed the SDL-specification to be incomplete, and the timers not optimally tuned; but the main bottleneck came from the runtime-system. Removing all three problems was time consuming but allowed to increase the throughput by more than two orders of magnitude [49].
2. The dynamic timing behavior of an industrial pick-and-place robot with parallel work and computer control was measured, modeled and carefully analyzed and optimized. Considering the load profile, correlation between tasks and function schedule, the throughput could be improved by about 25% [71,70], cf. also [30].
3. The efficiency of an industrial broadband-ISDN prototype station could be more than doubled [66].
4. The data granularity of landscape contour lines processed by a multiprocessor could be varied leading to an optimal speed-up [47].

These are just four impressive examples reflecting our own experiences. Many more examples can be found in original papers and books.

Performance measurements and experiments on real systems are vital for both industry and academia. They show what a system really does, evaluate its

overall usefulness and give insight into its detailed behavior; measurements also stimulate PE-modeling directions and allow to assess our own theoretical doing. In this article we focus on modeling and analysis due to the goal of the summer school; note, however, that the picture is complete only if one considers both sides of the PE-methodology [20,43,45]. We also have to skip the synthesis part, i.e. the combination of analysis and optimization, since this would be, again, beyond our scope.

3 Resource-Sharing Systems vs. Real Time Systems

The manifold of transportation and processing systems may be split into two categories, real-time and resource-sharing systems. Examples, their main purpose, properties and process models are summarized in table 2. However, note that depending on the level of abstraction, the same technical system may be viewed in one case as a real-time system, in another as a resource sharing system. This is particularly true for communication networks. They have to solve real-time problems but they share most resources due to economic reasons. Of major importance for us is -depending on the considered features- the completely different timing model of both classes:

1. Real-time systems need deterministic timing models because actions take place at distinct time instants or within fixed time intervals. Typical models are (time) extended finite state machines, timed automata, timed Petri nets, timed process algebras, and the like.
2. Resource-sharing systems need stochastic timing models due to contention, faults, mass phenomena, random service strategies etc. Randomly varying time instants and time intervals are captured by queuing models and stochastic versions of Petri nets, automata, graphs or process algebras.

We focus on resource-sharing systems and their modeling due to the goal of the summer school, being aware, however, that real-time system models include quite some potential also for the modeling of resource-sharing systems.

4 Classical Modeling of Workload and Systems

4.1 Why Stochastic Modeling?

Due to economic and technical reasons many systems are resource-sharing systems: Ticket counters, taxi services, telephone networks, mainframe computers etc. However, sharing of resources often leads to conflicts because customers try to access them simultaneously or with some overlap. Moreover, varying service times, error situations, background work, etc. add to such congestion processes.

The complexity of the situations, influences and conditions makes it impossible to describe such phenomena by deterministic models. Complicated interrelations, the lack of detailed information and some basic indeterminacy in the physical world make such processes to appear random. Nevertheless, measurements

Table 2. Characterization of Different Transportation and Processing Systems

Transportation and Processing System	
Real-Time System	**Resource-Sharing System**
Examples – Process Control – Manufacturing Systems – Robots – Avionic Computer Systems,...	Examples – Time Sharing Computers, Mainframes – Client-Server Architectures – Telephone-/Data Comm/Systems – Production Lines with Work-Over,...
Main objectives – Correctness of Process-Interaction – Fault Tolerance	Main objectives – Economic Use of Resources – Fault Tolerance
Properties of Interest – Safety – Liveness	Properties of Interest – Throughput, Utilization – Loss-Probability, Delay
Deterministic Timing – Actions take place at distinct time instants or within fixed time intervals	Probabilistic Timing – Contention, Faults and Mass-Phenomenon lead to randomly varying time instants and intervals
Process Models – Extended Finite State Machine (EFSM), Timed Petri Net, Real-Time Process Algebra, Timed Automata	Process Models – Queuing-Networks, Stochastic Petri-Net (SPN), Stochastic Process Algebra (SPA)

show that, although individual behaviors and different events are unpredictable, many statistical regularities can be observed and modeled by means of stochastic processes. Theoretical considerations of limiting behavior support these observations. Figure 4 summarizes some typical examples of values which vary unpredictably. Figure 5 shows some famous early experiments and performance predictions using stochastic models and assuring their results by measurements.

4.2 Basic Concepts

The dynamic behavior of resource sharing systems can be modeled by some fundamental concepts of statistics and probability theory. We briefly define and interprete the most important terms: random variables, distribution functions

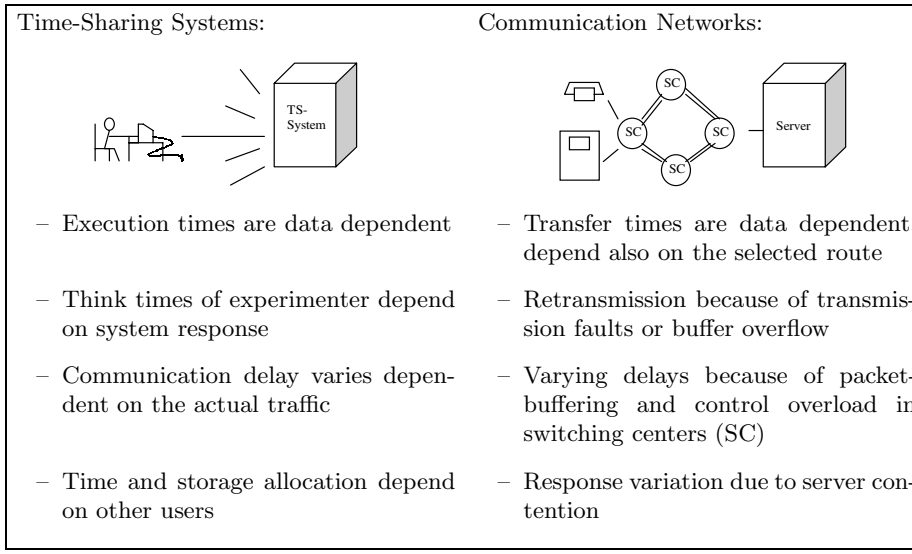


Fig. 4. Typical Examples for Randomness in Time-Sharing Systems and Communication Networks.

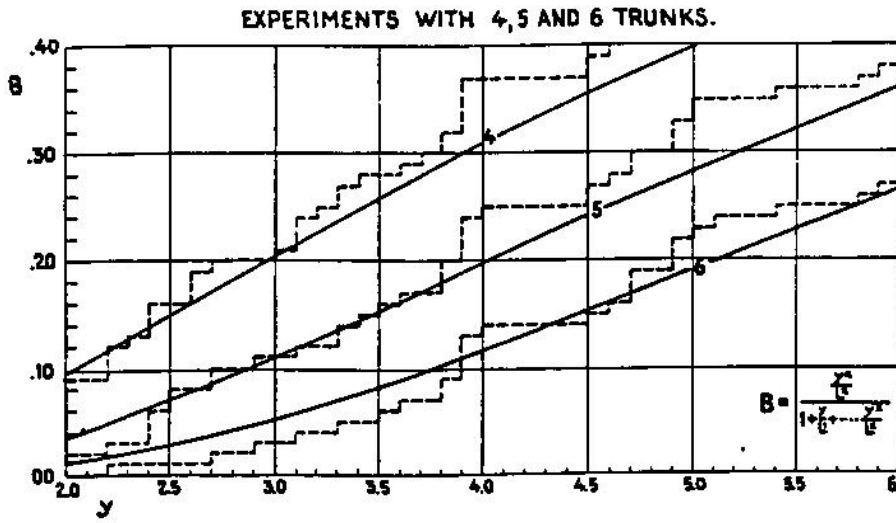
and stochastic processes. The uncertain outcome of an experiment or observation is captured by a random variable. Random variables are characterized and distinguished by their distribution function. Furthermore, a stochastic process allows one to describe a sequence of related experiments. The class of Markov processes is of special interest to us. All these concepts are summarized next, for more details cf. e.g. [46].

Definition 1. (**Random Variable**)
A random variable is a variable whose value depends upon the outcome of a random experiment. The axiomatic definition assumes a probability space $[S, \mathcal{A}, P]$, that is, a sample space S, a σ-algebra $\mathcal{A}$ of subsets of S, and a probability assignment P to the events of the sample space. Then a function $X : S \rightarrow \mathbb{R}$ is called a random variable because it assigns to each sample point $s \in S$ a real number $X(s)$. □

We distinguish discrete and continuous random variables dependent on their range, which may be countable or non-denumerable. Some examples are shown in table 3, cf. also Figure 4. The set of possible values (or states) that the random variable may take is called its state space E.

Definition 2. (**Distribution Function**)
The distribution function (or more precisely the probability distribution function, d.f. for short) F_X of a random variable X is defined to be the function

$$F_X(x) = P(X \leq x). \tag{1}$$



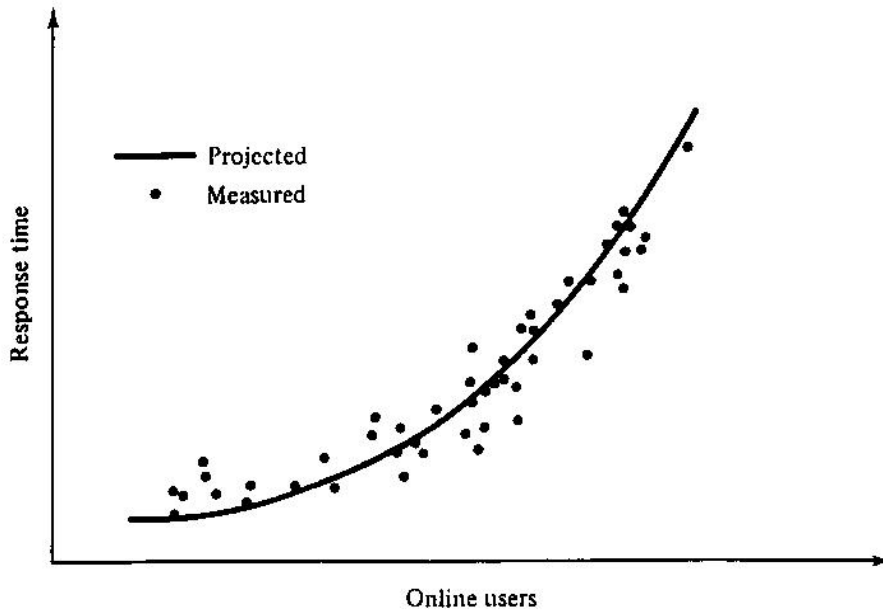


Fig. 5. Comparison of Measured and Predicted Performance Values. Early Results for Telephone Networks [16] (B probability of Loss, Y traffic) and Time-Sharing Systems [62].

Table 3. Some Examples for Random Variables (RV)

Continuous RV	Discrete RV
- Interarrival times of jobs	- Number of buffered jobs
- Activity times	- Idle/busy/overflow-states
- Waiting times	- Arrivals in a fixed interval

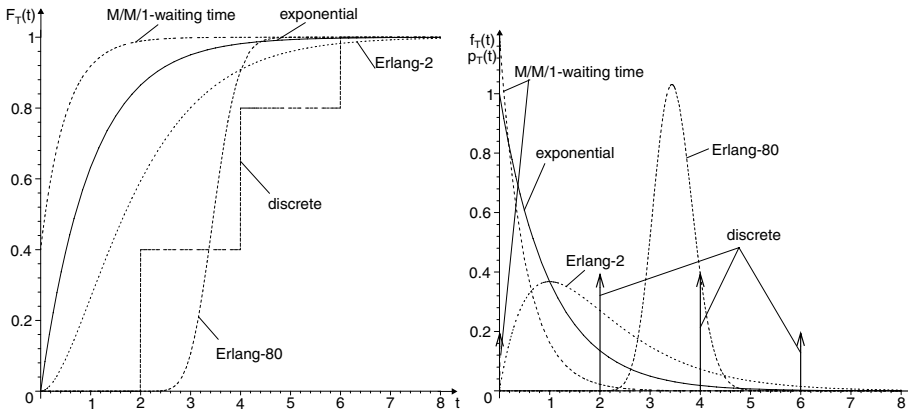


Fig. 6. Typical Examples for Distribution Functions $F_T(t)$ and their Related Density Functions $f_T(t)$ or Discrete State Probabilities $p_T(t)$: Exponential d.f., Erlangian d.f. of low or high order, discrete time d.f. and a typical d.f. for waiting times (here from a M/M/1-queuing station).

In case of a discrete random variable we have

$$F_X(x_n) = \sum_{i=0}^{n} P(X = x_i) \tag{2}$$

where $P(X = x_i)$ is the probability that X will have value x_i. In case of continuous random variables we get

$$F_X(x) = \int_0^x f(t)dt \tag{3}$$

where the function $f(x)$, the derivative of $F_X(x)$, is called the probability density function (or simply the density function) for the random variable X. □

Some typical examples for distribution functions and related probabilities or density functions are shown in Figure 6.

Definition 3. (**Stochastic Process**)
A stochastic process is a family $\{X(t)\}$ of random variables indexed by a parameter $t \in I$ and taking values of some state space E. Usually, t has the meaning

of time, I is some time interval, and the state space denotes the set of possible values (or states) of $X(t)$. □

The classification of stochastic processes considers both the type of random variables and the time instants at which state changes may take place. Some examples are shown in Figure 6. Note that discrete state processes are often called stochastic chains.

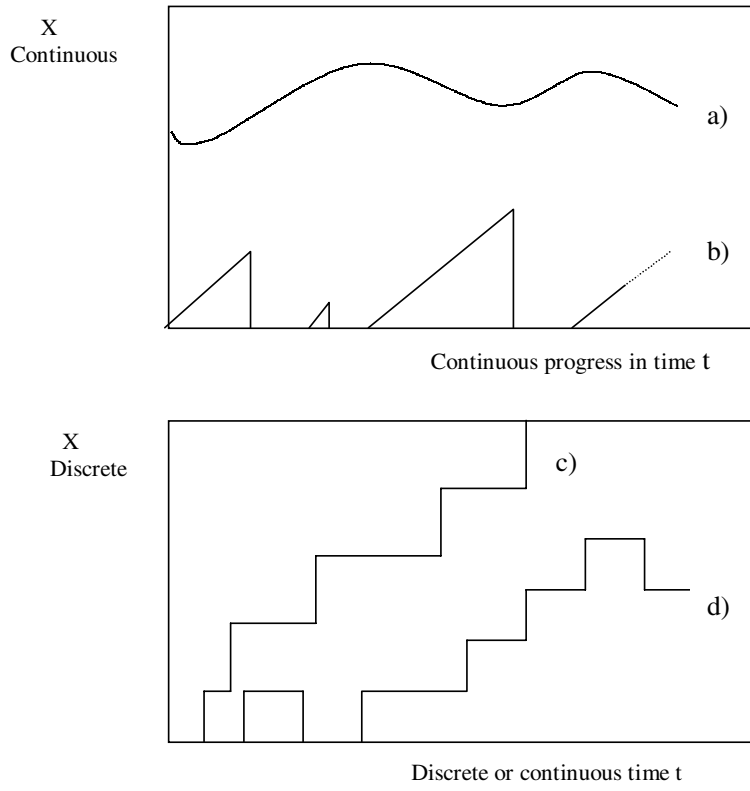


Fig. 7. Stochastic Process Examples: Mean packet delay in the Internet (a), duration of a telephone call (b), counting process (c), number of busy channels in an ATM-network (d).

The complete probabilistic description of an arbitrary stochastic process is not feasible. However, based on binomial trials there are two important limiting processes, Gaussian and Markovian processes. While the well known Gaussian process allows to accurately model many natural phenomena exposed to a large number of random influences, the Markovian processes are often well suited for

our purpose i.e. the modeling of resource sharing system dynamics (a refined classification can be found in Figure 13).

Definition 4. (**Markov Process**)
A stochastic process is called Markov process if its future evolution depends only upon its current state and not upon any previous values. More formally, the conditional probabilities are given by the following relation:

$$P(X(t) = x | X(t_1) = x_1, X(t_2) = x_2, \ldots, X(t_n) = x_n) = \\ P(X(t) = x | X(t_n) = x_n) \tag{4}$$

for arbitrary $n \in \mathbb{N}; t_1, t_2, \ldots, t_n, t \in I$ with $t_1 < \ldots < t_n < t$ and $x, x_1, \ldots, x_n$ included in some state space[2]. Markov processes with discrete state space are called Markov chains (MC). Dependent on their timing behavior we talk about CTMC (Continuous Time MC; changes in state may occur anywhere within a (set of) finite or infinite intervals in the time axis) or DTMC (Discrete Time MC; changes in state may take place at time instants which are finite or countable). □

Markovian processes

- posses per definition an outstanding mathematical property, the memoryless or Markov property
- have a solid mathematical foundation laid by researchers like Einstein (1905), Markov (1907), Erlang (1909/18) and Kolmorgoroff (1931),
- are very well investigated and many analytic results as well as efficient numeric analysis techniques are known, and
- often approximate measured system dynamics in nature and society very well.

4.3 The Family of Exponential Distribution Functions

From books on statistics and probability, various distribution functions for random variables are well known, e.g. Gauss-, Gamma-, geometric- and the Weibull distribution. Some of them are well suited to precisely describe system dynamics such as interarrival and service times; however, to systematically derive performance characteristics is extremely difficult or impossible. To avoid these evaluation problems a special family of distribution functions is mostly used: It is based on the exponential d.f., the only continuous d.f. which fulfills the memoryless property of Markovian processes and allows one - as we will see - a systematic and relatively simple analysis of models, i.e. performance values may be derived with relative ease. Moreover, an arbitrarily close approximation of any non-negative random variable is possible by the superposition of exponentials.

We briefly characterize the exponential d.f. as a prerequisite for the next section on Markov chain analysis. Some remarks on the family and examples conclude this section.

[2] If the permitted times at which changes in state may take place are finite or countable we often write X_t rather than $X(t)$.

Definition 5. (**Exponential Distribution**)
The distribution and density functions as well as the expected value and the variance of the exponential distribution are given by the following relations

$$F_X(x) = P(X \leq x) = 1 - e^{-\lambda x} \quad ; \quad x \geq 0 \tag{5}$$

$$f_X(x) = \frac{d}{dx} F_X(x) = \lambda \cdot e^{-\lambda x} \tag{6}$$

$$E[X] = \int_0^\infty t \cdot f_X(t) dt = \frac{1}{\lambda} \tag{7}$$

$$VAR[X] = \int_0^\infty (t - E[X])^2 \cdot f_X(t) dt = \frac{1}{\lambda^2} \tag{8}$$

□

As mentioned before, it is the only continuous d.f. for which the memoryless property holds, it has a constant transition rate λ irrespective of the past behavior and its residual sojourn time is a random variable with the same density function as the whole sojourn time

$$f_X(x + d | x \geq d) = f_X(x) \tag{9}$$

A proof of these unique features can be found in the standard literature, e.g. in [46]. The density function $f_X(x)$ is shown in Figure 8 beside other functions.

Definition 6. (**Family of Exponentials**)
A family of exponential distribution functions is defined by the serial superposition of exponential distribution functions and by the probabilistic selection out of different branches of exponentials or series of exponentials. □

This superposition of phases can be described, interpreted and analyzed as a multidimensional Markov process.

Figure 8 shows the density functions for some members out of this family. Numerous experiments and measurements show that the exponential d.f. often allows one to accurately model real situations in road traffic, in telephone and data networks or time-sharing and mainframe computer systems. Using superposition of exponentials even a much wider class of real situations can be modeled; then, however, we often are faced with the so called state-space explosion problem, i.e. the state-space grows dramatically and compute times for the numerical evaluations may explode. This problem will be mentioned later on, again.

4.4 Representation of Continuous Time Markov Chains

Recall that a Continuous Time Markov Chain (CTMC) is given by discrete states and exponential residence time in each of them, cf. Definition 4 and 5. The standard representation of such Markov chains is given by state transition diagrams, suited for graphical representation, or by a generator matrix, suited for

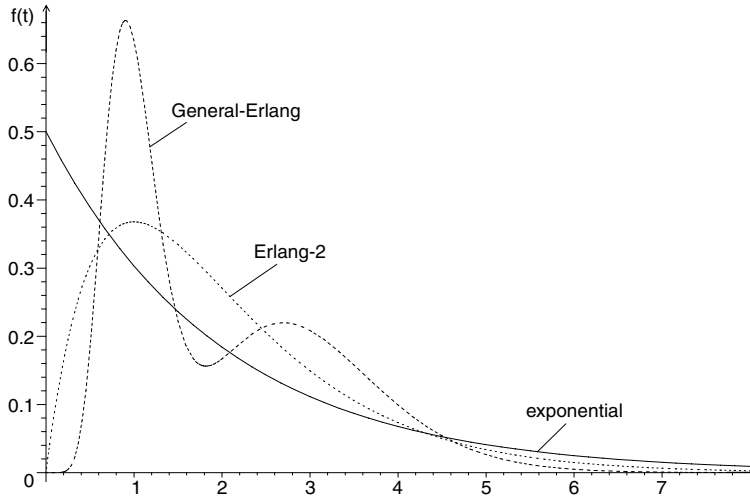


Fig. 8. Density Function Examples for Different Markov Processes: Exponential-, Erlang-k- and General Erlangian Distribution Function

computations. State transition diagrams show the set of possible states and the transition rates between them. The generator matrix is the corresponding matrix representation; for an efficient evaluation it stores, however, in the diagonals the negative row sum ($\sum$) of transition probabilities. Figure 9 shows some simple examples.

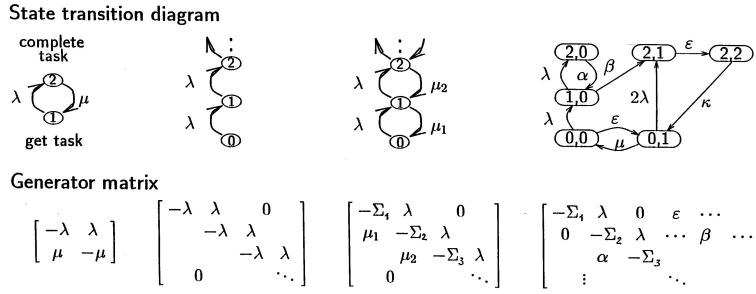


Fig. 9. Some Examples for the Standard Representation of Markov Chains, cf. Text.

The uniqueness of processes with Markovian behavior is captured by the so called Chapman-Kolmogoroff system of equations for the transition probabilities. Since we are mainly interested in the unconditioned state probabilities $P_j(x)$ some transformations allow one to derive a system of differential equations

$$\frac{dP_j(t)}{dt} = \sum_{i \neq j} P_i(t) \cdot q_{i,j}(t) - P_j(t) \cdot q_{j,\cdot}(t) \qquad j \in E \tag{10}$$

where the transition rates $q_{i,j}(t)$ are usually independent of t, and $q_{j,\cdot}(t)$ represents the negative row sum $-\sum_j$. The interpretation of these equations is straightforward: The differential change of the state probability $P_j(t)$ corresponds to the difference between its probability of emergence from other states and its probability of disappearance to neighboring states at time t.

The transient solution is very meaningful when the system under investigation needs to be evaluated with respect to its short term behavior. Considering it for the long run, however, it can be shown that the state probabilities often converge to constant values. These steady-state- or equilibrium equations can be readily derived from the above system of equations. One gets

$$P_j \sum_{\substack{k \\ k \neq j}} q_{j,k} = \sum_{\substack{k \\ k \neq j}} P_k \cdot q_{k,j} \tag{11}$$

This system of equations expresses that the disappearance of a state j to other states and its emergence from there are in a statistical equilibrium.

Markov processes have the invaluable advantage that we always know - at least in principle - how to investigate them. The standard solution technique includes the following steps:

1. Define all states j of the state space E and determine the corresponding transition rates $q_{j,k}$
2. Determine the Chapman-Kolmogoroff differential equations system or, for stationarity, the linear equilibrium equations system.
3. Compute the state probabilities in either case analytically or numerically.
4. Derive standard performance measures - throughput, waiting times, etc. - from the state probabilities P_i.

More details and literature can be found in section 5.

4.5 Remarks on System Modeling

Up to now, we assumed that the state-transition diagram, representing all states and possible state transitions, is given. However, its structure and parameters heavily depend on the system under investigation; moreover there is not a unique system representation and skillful modeling and state-definitions may ease the evaluation process.

Here, we only show some basic system models which can be used in many different situations and sketch the idea of modeling of complex systems.

Basic System Models The single-server-queue is one of the most exciting models allowing one to describe and investigate many different situations. Figure 10 shows a self-explaining scene, different interpretations in technical systems

and the usual symbolic representation. Note that the model is very abstract with many implicit assumptions (scheduling strategy, priorities, type of arrival/service processes etc.). The lower part shows the corresponding state-transition diagram if we assume both exponential arrivals and service times.

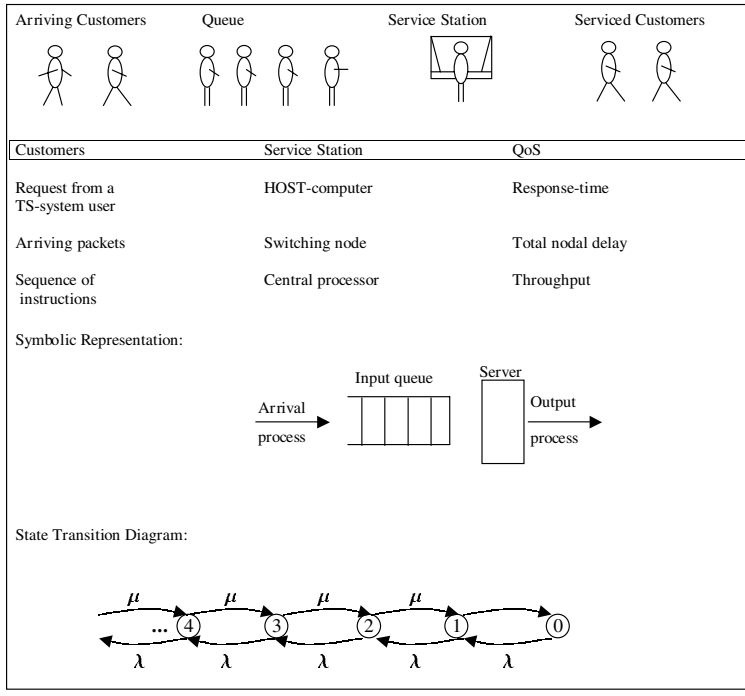


Fig. 10. The Single-Server System. (Abbreviations: TS-time sharing, QoS-quality of service)

Many performance results have been derived, evaluated, tabulated or programmed in tools for the single server system. E.g. the pioneer J. Cohen (who passed away only recently) investigated this fundamental model carefully on more than six hundred pages [14].

Another example for an important class of queuing model is the central server model with a constant number of customers. A simple instance of such a model is shown in Figure 11. Buzen [11] developed this class of models to analyze the performance of computer systems with a single processor - the central server - and various I/0 devices; the limited number of customers reflects the fact of a limited (fixed) degree of multiprogramming.

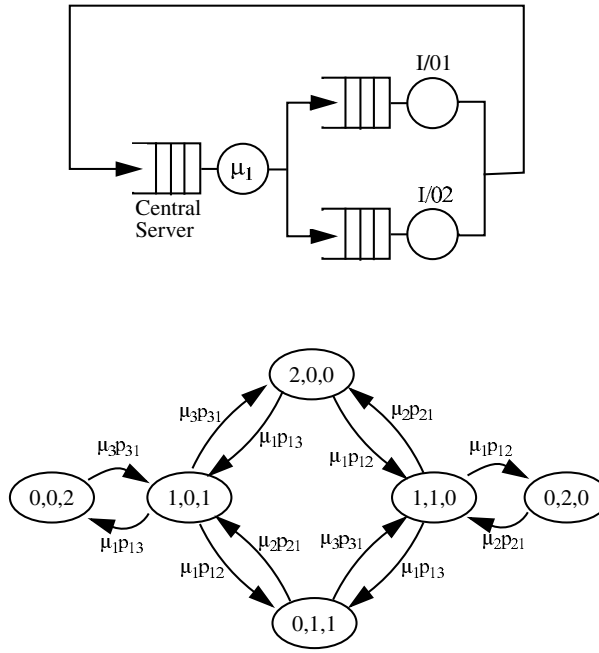


Fig. 11. A Central Server System. Closed Model with two I/O-stations as well as state-transition diagram in case of two customers ($K = 2$) exponential service (service rates $(\mu_i, i = 1, 2, 3)$ and transition probabilities p_{ij} from queue i to queue j.

Hierarchical Modeling Suppose we have to investigate the performance of some host-to-host communication in a national or international network with virtual connections. Trying to model such a complex scenario with a single model is impossible: it either does not represent the detailed system structure and dynamics accurately or it is too unwieldy for performance evaluation and result validation.

A very successful approach in such situations is the hierarchical modeling technique. Its basic idea is the stepwise refinement of complex scenario models. Parts of the macro-level model are detailed by intermediate-level models which may be again refined into micro-level models, etc. The modeling steps and the corresponding modeling hierarchy for our initial host-to-host communication scenario are shown in Figure 12:

At first, we have to study the fundamental sequence of events in the real system: connection request, successful and unsuccessful establishment of a virtual connection, in case of success data transmission. The corresponding queuing model is shown next: a single-server with two phases of service and a feedback loop; t_1 represents the time of trying to establish a connection, q is the probability

for an unsuccessful trial while t_2 describes the transmission time. At least, t_1 and q are unknown, often all three parameters. In each case, intermediate models - representing the signaling procedure as well as switching and transmission in the network - have to be built.

These intermediate models may be refined, again, modeling congestion in the control units and blocking in the switching units individually for each node of the network. [3]. In analyzing this model hierarchy, these micro-level parameters are fed into the intermediate-level models allowing one to determine the three unknown parameters of the macro-level model. Finally, the macro-level model enables us to estimate the wanted overall-delay dependent on cross traffic, routing strategies, network topology etc.

This procedure of top down modeling and bottom up analysis has been often and very successfully used since the 1970ies [48] in modeling computer and communication systems. Layered queuing networks for example are a popular technique supported by elaborate tools [72,58,41]. However, in general, the decomposition strategy, the selection of models and especially the interfacing of various models is an art mastered only by very experienced specialists. Beside the basic idea - stepwise refinement - there are usually no rules, no guidelines supporting the modeling and evaluation procedure. However, in section 6 on advanced modeling techniques we will see some first steps for a systematic and general solution.

5 Analysis of System Dynamics

5.1 General Remarks

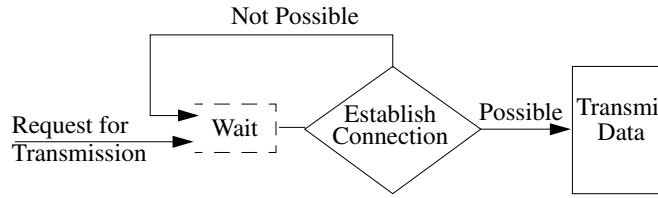
Up to now, we have given a justification for stochastic modeling, we have surveyed its theoretical foundation and we have seen some typical examples for classical models. The question is now how to analyze the time-dependent dynamic system behavior and how to determine characteristic QoS values? There are various performance measures we may be interested in. Dependent on our role or view we may need qualitative statements representing a system's usefulness for customers or its economical value for the operating authority. Typical examples are

- User view: Waiting time, response (sojourn) time, total delay (in all cases mean values, variance and distribution function).
- System view: Utilization and throughput of different resources, number of waiting customers at various locations, probability of overflow in case of alternate or adaptive routing, etc.

Independent of the qualitative statements in which we are interested, we can distinguish three classes of evaluation techniques:

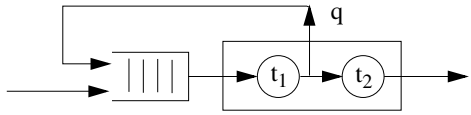
[3] Technical details on these submodels may be found in [27] and the standard literature

1. Study the Basic Sequence of Events



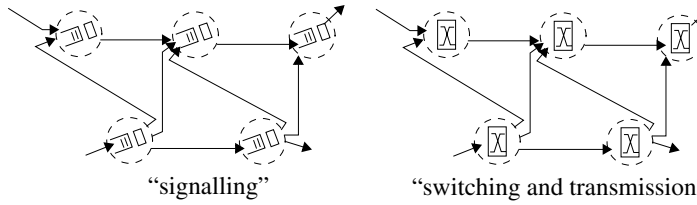
2. Develop the Corresponding Model

- Macro-Level Model



3. Study the Details of the Sequence of Events

- Intermediate-Level Models



- Micro-Level Models

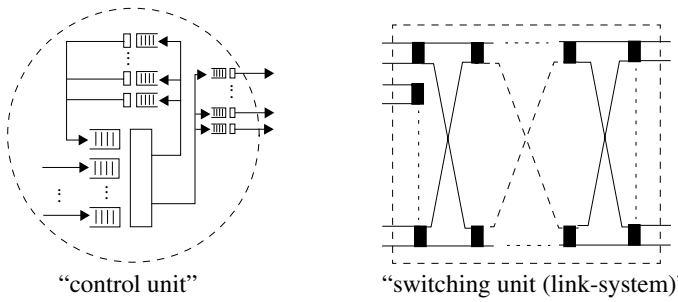


Fig. 12. The Process of Hierarchical Modeling, a Case Study [27].

– Analytic/Algebraic Methods
– Numerical Analysis, and
– Simulation

They are characterized and outlined in the following three sections. These summaries also point to relevant literature and contributions in this book.

5.2 Analytic/Algebraic Methods

When we speak of analytic/algebraic models we usually mean a solution technique that allows us to write a functional relation between system parameters and a chosen performance criterion in terms of equations that are analytically/algebraically solvable [48]. The theory of service systems and particularly queuing theory provides a mathematical framework for formulating and solving such problems.

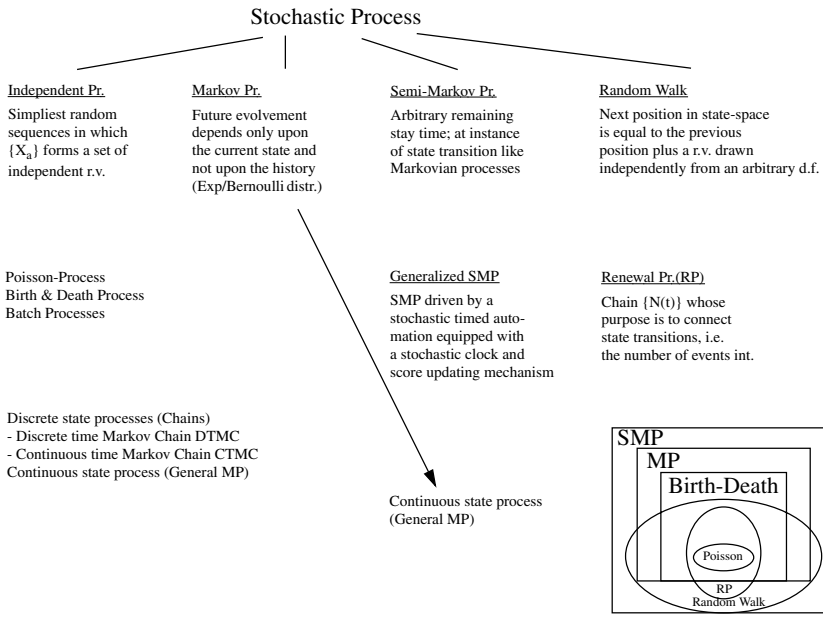


Fig. 13. Classification of Stochastic Processes.

There is a variety of methods dependent on the type of stochastic behavior by which the system can be described with. Figure 13 classifies stochastic processes and relates them to each other.

The Chapman-Kolmogoroff system of equations and the steady-state equations, cf. section 4.4 are often the base of performance analysis; event sequence

State-Based Approaches	Event Sequence Approach
Principle: Use Chapman-Kolmogorov or Steady-State Equations Examples: – Standard Approaches (MP) – Phase Concept and Matrix Analytic Method – Embedded Renewal Processes – Supplementary Variables – Diffusion-Approximation, ...	Principle: Explore special sequences of events Examples: – Lindley's Integral Equations – Method of Moments – Convolution of Exponentials – Piecewise Exponential Functions – (Max,+)-Algebra, ...
Bounding Techniques	**Fundamental Laws**
Principle: Consider the process under special conditions Examples: – Asymptotic Bounds – Balanced System Bounds, ...	Examples: – Little's Law – Flow Equivalence Law – Work Conservation Law, ...
Transformation Techniques	**Special Network Algorithms**
Principle: Change the form of an equation to one which is easier to investigate Examples: – Laplace-Transform – Generating Functions – Discrete Fourier-Transform – Complex Cepstrum, ...	Principle: Utilize the special structure and system parameters Examples: – Jackson's State Probabilities – Buzen's Convolution Algorithm – Mean-Value-Analysis of Lavenberg and Reiser, ...

Fig. 14. Elements of Analytic/Algebraic Solution Techniques

analysis is another general principle and many different solution techniques have been proposed. Both principles and other successful elements of analytic/algebraic solution techniques are summarized in Figure 14. Books like [59,46,48,69,22,68] survey the various techniques and lead to original papers. Some very sophisticated techniques are also presented in this book, e.g. by German [21] and Ciardo [13].

5.3 Numerical Analysis

Closed form analytic or efficient recursive solutions are possible only for a few classes of models. For the general case, however, such direct techniques are not available. Nevertheless, a wide variety of standard and customized numeric tech-

niques are at hand. Stewart [64,65] proposes the following three step methodology:

1. Describe the system to be analyzed as a Markov chain.
2. Determine a matrix of transition rates and transition probabilities.
3. Numerically compute performance measures of interest.

Figure 15 summarizes the different numerical methods. For more details see [64] or [6] and Haverkort's [26] and Ciardo's [13] contribution in this book.

Direct Methods	Iterative Methods
Principle: Operate and modify matrix – Fill-in effect of matrix entries – Accumulation of round-off errors Examples: – Gaussian elimination – Grassmann, Taksar, Heyman variant Reliable and accurate for small models with some hundred or thousand states	Principle: Preservation of matrix sparsity, successive consequence to the solution – Convergence not always guaranteed – Sensitive to the parameter values and the initial estimate Examples: – Power method – Jacobi, Gauss-Seidel, SOR

Other Methods
- Projection (create vector subspace)
- Recursive (stability problem)
- Matrix geometric (certain Markov chains)
- Uniformization (randomization, Jensen, transient)

Fig. 15. Successful Methods for Numerical Analysis

5.4 Simulation

Kobayashi [48] describes the role of analytic models and simulation to the point: "An analytic model should be sought wherever possible, since it can evaluate the performance with minimal efforts and cost over a wide range of choices in the system parameters and configurations. Even with simplifying assumptions and decompositions, however, the resultant analytic model is often not mathematically tractable. Then the only alternative for predicting the performance of a non existing system is a simulation."

In simulation, objects, their properties and relations are described by some data structure whereas their specific behavior, interactions and evolution in time is formulated by a program. There are several disadvantages of simulation, summarized in the following list:

– Complicated programs that execute a large number of trials. Not easy to debug.

- Very expensive to run for reliable computer experiments (e.g. the simulation of one second real-time of an ATM-ring configuration took us four hours of simulation time on a SPARC).
- Real optimization at reasonable expense is almost impossible.

But simulation is necessary and advantageous in many situations:

- A detailed and flexible modeling and evaluation of arbitrary system dynamics including sophisticated interdependencies is possible .
- First estimates for new types of problems may be obtained readily.
- Comfortable simulation tools are available, easy to understand and therefore highly accepted for performance assurance.

The pros and cons are detailed in standard books, e.g. [54,43] and in Cassandras' book [12]. But from the above we already can see that simulation and analytic methods complement each other.

6 Advanced Integrated Approaches

6.1 Deficiencies of Classical Modeling and Analysis Technique

Classical modeling and analysis techniques have a high standard, are very elaborate and many fundamental results as well as efficient PE-tools are available. In many situations, even for complex hard- and software structures, they allow to accurately predict system performance. This is particularly true when engineers and performance specialists work closely together.

However, this is not the normal situation especially when software-engineers are involved as set out in section one. Moreover, classical workload assumptions take for granted that (except for simulation)

- processes occurring at the same time are independent of each other (e.g. simultaneous telephone calls or competing user programs), and
- processes being dependent on each other take a sequential turn (e.g. the sequences *connection*→*data*→*acknowledgment* or *input*→*processing*→*output*).

Today, however, modern multiprocessor-systems, workstations organized as a cluster or distributed systems work differently in general. This means that they

- take advantage of the parallelism inherent in many control and application tasks, i.e.
- tasks are decomposed into well-defined cooperating subtasks, competing for the same resources, and serviced in parallel or with some overlap.

In all these situations, we have to be aware of the simultaneity and extreme interdependency. A simple example with two tasks arriving simultaneously and processed by two servers is shown in Figure 16.

- If the tasks are independent of each other the expected response time is 1.4 seconds and the shape of the overall d.f. is the same as for the individual tasks, i.e. the d.f. does not change.

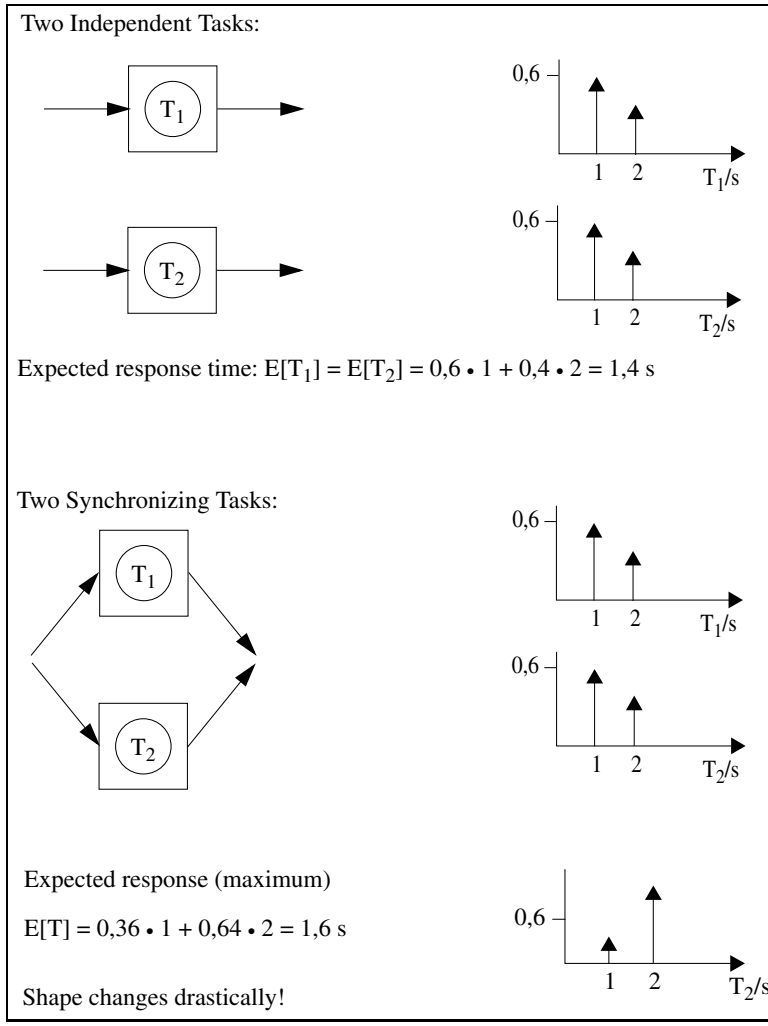


Fig. 16. Response Time in Case of Task Synchronization.

– If we have synchronizing tasks, i.e. service is completed if and only if both tasks are processed, the expected response time amounts to 1.64 seconds; but even more important is that the shape of the d.f. changes drastically.

Again, this is only a very simple example. There are extensions of classical models ("synchronizing queuing models") taking into account this effect, but only for very simple dependency structures and not for the general case.

Working in the area of multiprocessor-performance modeling we started already in the 1980ies to extend classical queuing models but the expressiveness is very limited. From our experience and point of view, only integrated approaches,

i.e. approaches considering at the same time functional and temporal aspects, offer a general, flexible solution.

Such integrated approaches allow one to capture the above mentioned effects in a unified system description. And very important, such descriptions might allow us to enable the areas of SW and HW development to enjoy the benefits of PE in a more or less automatic fashion. All kind of scenarios are imaginable and under investigation with the common goal of avoiding typical performance bottlenecks in system design: There is research from

- the open specification and analysis technique showing both the functional and temporal behavior and QoS-parameters of a system, e.g. [31], over
- the automatic instrumentation and measurement of an implemented prototype [49], up to
- the specification and implementation technique which completely hides performance details, but gets performance parameters out of a library, and composes hardware- and software components in an optimal way [63] [19].

In the following, we briefly survey three integrated approaches which are promising and described in more detail by separate contributions of Balbo [3] and Sanders [61] or Brinksma and Hermanns [7] and Katoen [44]: Stochastic Petri nets, stochastic activity networks and stochastic process algebras. The length of these summaries reflects our own experience and involvement in these advanced integrated modeling techniques.

6.2 Stochastic Petri Nets

Petri nets (PN) are an early and effective modeling tool for the description and analysis of concurrency and synchronization in parallel systems exhibiting the cooperative actions of different entities [1]. They were introduced by C.A. Petri [56], thoroughly investigated during the last thirty years and extended to capture also performance issues [53,1].

The structure of a standard PN is a bipartite graph consisting of two types of nodes N, places P and transitions T - and a set of directed arcs A between them. In a graphical representation of PNs, places are drawn as circles and transitions as bars, cf. Figure 17. The dynamic behavior of a PN is modeled by tokens marking activated places and firing rules:

- the transitions are enabled when all input places contain at least one token, and
- enabled transitions can fire, i.e. remove one token from each input place and put one token to each output place.

Thus, the state of a marked PN is defined by the number of tokens contained in each place. The modeling of a queuing situation where customers arrive, may wait in the queue Q for the server S to be free, then being serviced and departing from the system by means of a PN is illustrated in Figure 17.

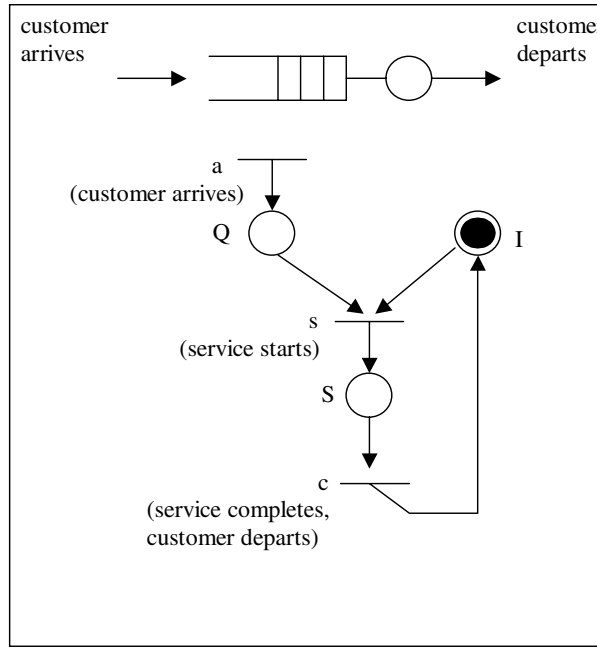


Fig. 17. A Queuing Situation and its Functionally Detailed PN-Representation.

Functional properties, such as deadlock situations or safety may be analyzed by the so called reachability analysis. If we associate with each or some transitions in a PN an exponentially distributed r.v. that expresses the delay from the enabling to the firing of the transition we obtain a SPN - Stochastic Petri Net - or GSPN - Generalized SPN - , respectively.

In these cases firing rates need to be specified, which can be marking dependent. Now, both functional and all kind of temporal properties can be analyzed. Standard text books, e.g. [1,8,6] as well as Balbo's [3] contribution carefully introduce Stochastic PN and survey the state-of-the-art. From our point of view, we summarize the pros and cons of SPN as follows

Advantage

- Mature in both theoretical exploration and tool development
- Performance bounds derivable already from PN-structure, detailed performance analysis by mapping the marked PN onto a Markov chain
- Successfully applied in many situations where synchronization has a determining influence, also in industrial studies

Disadvantage

- Expenditure for the specification and representation of large systems very high (Gorrieri: exponential compared to linear in case of process algebras [25])

– Parallelism inherent in the marked PN is not reflected in the underlying Markov chain.

6.3 Stochastic Activity Networks

Stochastic Activity networks (SAN) are a particular class of Stochastic Petri Nets. In order to support an efficient modeling of systems they provide the following building blocks:

– Places
– Activities (exponential, constant or instantaneous transitions),
– Input Gates, controlling the enabling of activities dependent on some input places (Boolean enabling predicate and input gate function)
– Case Probabilities (possible outcomes of an activity, dependent on the network marking)
– Output Gates, defining the marking changes of a single output.
– The operations "Join" and "Replicate" for constructing modular models.

With the help of these building blocks, SANs allow one to describe a system's structure and dynamic behavior in a very compact way. This compactness, of course, comes at the price that understanding a model is sometimes not so easy.

Figure 18 shows the SAN model of a faulty microprocessor. SANs are tailored for the modeling and evaluation of large-scale systems with many symmetric subcomponents. Taking into consideration such symmetries, enormous state-space-reductions are possible. QoS-Parameters can be derived using the concept of impulse rewards. From our point of view, the main advantage of SANs is their ability to construct models in a modular fashion with the help of the "Join" and " Replicate" operators, and that in the presence of "Replicate" an automatic reduction of the state space is carried out. More details and a state-of-the-art summary can be found in the contribution of Sanders [61].

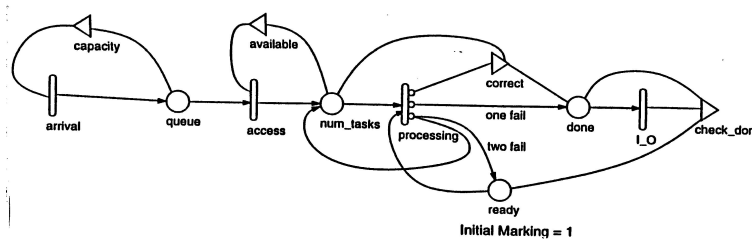


Fig. 18. SAN Model of a Faulty Microprocessor [60].

6.4 Stochastic Process Algebra

Being confronted with all kinds of synchronization and cooperation problems of an experimental, hierarchical multiprocessor [18] we started already in the late eighties to study process algebras and to embed stochastic processes. The following is a excerpt from some recent publications and summarizes the main features, and the current state-of-the-art of process algebras [31,29,30,41].

The Concept Classical process algebras, among them CCS [52], CSP [40], or LOTOS [4], have been designed as formal description techniques for concurrent systems. Therefore, they are well suited to describe the functionality of reactive systems, like operating systems, automation systems, hierarchies of communication protocols, etc.

From the very beginning, the basic idea of process algebras was to systematically construct complex systems from smaller building blocks and to check formally whether systems behave equivalently or not.

The behavior of each building block - whether hardware, software or a combination of both - is described as a process which may communicate with other processes. Standard operators allow various kinds of process composition, synchronization and communication. Therefore, software processes can be mapped onto other, more elementary software processes - this may be done repeatedly - and finally mapped and executed on computer- or communication hardware. Such systems can be combined again to build a network, etc. Since such network systems are very complex, another important operator allows one to abstract from internal details at any level of system description.

A process algebra can be seen as a progressive extension of classical automata theory enhanced by the above sketched features. However, instead of describing the state transition system directly a two step methodology is used (cf. Figure 19):

- The system is described with the help of a high-level language which is quite user-friendly and design-oriented (our input language is an enhanced version of BASIC LOTOS-Language of Temporal Ordering Specification-, the core language of ISO-standard 8807).
- A rule system called formal semantics allows one to automatically translate the language expressions into states and transitions of the labeled transition system (semantic model).

Two systems or system components are considered equivalent if the transition systems show the same (functional) behavior. There are several possibilities to define this formally: Trace equivalence means that all possible sequences of actions are identical. Popular is also bisimulation, meaning that both systems (or components) simulate each other in any situation. Having defined equivalent behavior, equational laws - deduced by axiomatization - reflect these equivalences on the system description level; therefore, comparison of two systems or system components is possible on both levels and therefore, one speaks of process algebras.

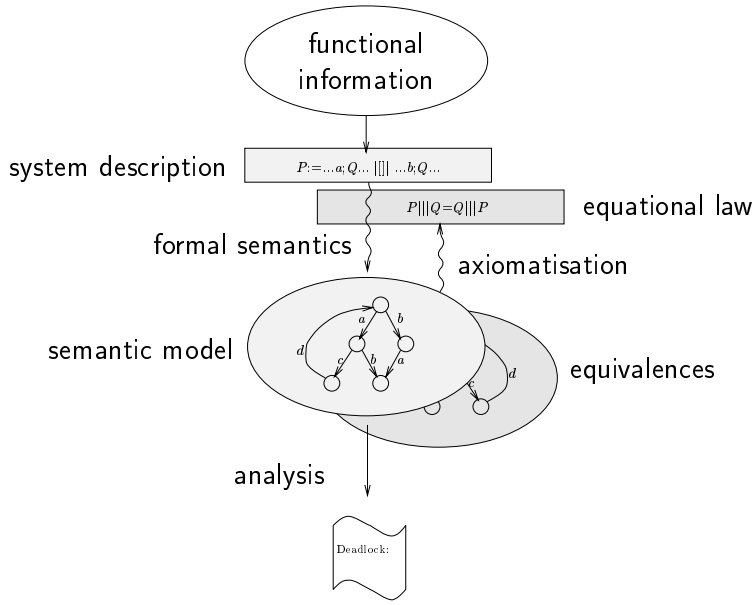


Fig. 19. Basic Concept of Process Algebras. In the case of stochastic process algebras, temporal information is added at each level and analysis includes performance measures as well as mixed properties (cf. section 6.4).

Classical process algebras dealt exclusively with the functional aspects while features of performance evaluation have been added during the last decade, cf. [23,32,51,10].

Stochastic Process Algebra The main motivation behind the development of stochastic process algebras has been to accurately describe and investigate the behavior of resource-sharing systems and to benefit from their unique properties also in case of performance modeling. To achieve this goal, temporal information has been attached to process descriptions in the form of (continuous) time random variables. These random variables allow one to represent time instants as well as durations of activities. Then, the concept of stochastic process algebras follows the lines of classical process algebras:

As before - cf. Figure 19 - the main ingredients are a formal mapping from the system description to a transition system and substantive notions of equivalence. Equational laws reflect these equivalence on the system description level. Rather than considering only the functional behavior we add stochastic timing information. This additional information in the semantic model allows the evaluation of various system aspects:

- functional behavior (e.g. liveness or deadlocks)
- temporal behavior (e.g. throughput, waiting times, reliability)

– combined properties (e.g. probability of timeout, duration of certain event sequences)

The stochastic process associated with every specification is the source for the derivation of performance results. Its characteristics clearly depend on the class of random distributions that are incorporated in the system description. Several attempts have been made to incorporate generally distributed random variables in the model [23,28,57,9]. However, the general approach suffers from the problem of efficient analysis techniques as well as general algebraic laws (except [15,44]). Therefore, usually exponential or phase-type distributions are embedded into the basic functional system description. Some simple examples of such process algebra descriptions with embedded exponential phases are shown next. (We directly relate the temporal behavior to individual activities as usually done; Hermanns showed (c.f. [7]) the advantage of an orthogonal approach).

– The sequential arrival of three different jobs is specified by a process *Jobstream*, describing explicitely each arrival point before halting:

$$Jobstream := (job_1, \lambda_1).(job_2, \lambda_2).(job_3, \lambda_3).Stop$$

– Consequently, a Poisson-arrival process is defined by an infinite sequence of incoming requests $(in, \lambda).(in, \lambda).(in, \lambda)\ldots$, which can be formulated recursively:

$$Poisson := (in, \lambda).Poisson$$

– A service process consisting of an Erlangian distribution of order two is given by:

$$Erl2 := (end_1, \mu).(end_2, \mu).Stop$$

– Both a precise and concise description of many service or arrival processes is possible; this is illustrated by a so-called train-process, which is important for the modeling of file transfers in local area networks. Thereby, the overlap and interleaving of different 'trains' is captured by the parallel operator (|||):

$$Train := (lok, \lambda).((wag_1, \mu).(wag_2, \mu)....(wag_n, \mu).Stop)|||Train$$

Mapping of software components onto other software modules or hardware is a very important modeling step. Let us consider a very simple example:

– There is a software job consisting of two independent tasks; their different functionality is expressed by different exponential times:

$$Job := start.(task_1, \lambda_1).Stop|||start.(task_2, \lambda_2).Stop$$

where ||| indicates again the parallel operator without synchronization. We also assume off-the-shelf processors the first of which ($Proc_1$) works at unit speed while processor $Proc_x$ is x-times as fast:

$$Proc_x := start.(task_i, x).Proc_x$$

– Mapping the job onto a specific processor is given by the parallel composition of both, taking into consideration that they have to synchronize on the events "$start$" and "$task_i$" (|(..)|-operator). Since we are not interested to "see" the timeless start signal we may hide it. Then the behavior of the complete hardware/software system is given by

$$System_x := hide\ start\ in\ (Job|(start, task_i)|Proc_x)$$

where the behavior of the Job and $Proc_x$ is given by the above descriptions.

These examples show how precise and concise such descriptions are. Of course,very general task and processor behaviors can be modeled and some demanding examples are mentioned when we talk about experiences.

From such a modular system description we automatically generate the labeled transition system; it contains all functional and temporal information. By hiding its functional information we directly derive the underlying CTMC.

The state-space explosion problem associated with the CTMC can be reduced significantly by compositional model generation and reduction. This is a very important feature treated in more detail by Hillston [33]. Another promising approach uses functionally correct and temporally approximate decomposition and reduction techniques; this permits the analysis of specific system models with a very large number of states [37,55].

Tool Support At the moment, there are three tools available, the PEPA Workbench [24], Two Towers [5] and our TIPPtool [38,34]. The TIPPtool is a prototype modeling tool which contains most of the specification and evaluation features of today's stochastic process algebras. Its main characteristics are:

– LOTOS oriented input language;
– Implementation of the structured operational semantics;
– Investigation of functional properties by reachability analysis;
– Analysis of temporal properties with various numerical solution modules for the transient as well as stationary analysis of continuous time Markov chains;
– User guided exploitation of symmetries and exact or approximate compositional reduction techniques; and,
– Computation of standard performance and reliability measures.

The tool is implemented in Standard ML and C and has an elaborate graphical interface. Medium size problems up to 100K states can be solved easily; very complex application problems with a certain structure may be solved by the approximate decomposition techniques [37,55] or via a transformation to stochastic graphs [28,50]. We also use our tool in combination with mature tools for purely functional specification, analysis and code generation.

Experiences There are about fifteen research groups dealing with stochastic process algebras. A complete theory and tools are available for models with Markovian assumptions including immediate transitions (actions consuming no

time compared to the others). Many small and medium size examples have shown the practicability of the stochastic process algebras concept. Recently, researchers began to solve large non-trivial problems, e.g.

- adaptive mechanism for packetized audio over the Internet
- a Manhattan style mobile communication system
- the Erlangen Hospital Communication System
- a plain-old telephone system (POTS)

While the first two examples model systems with generally or constant distributed time intervals and evaluate them by simulation [5], [2] the others embed exponential and phase type distributions and use the PEPA-Workbench [24] or the compositional model construction and analytic techniques of the TIPPtool [38,29].

The main disadvantage of SPA is that they are not yet completely developed; acceptance is also still low because of their unconventional theoretical foundation. But SPA introduce unique features into PE: The abstraction process for complex system modeling is supported and the state-space can be reduced automatically using algorithms checking for equivalence. Moreover, stepwise composition including state-space reductions allows further state-space savings.

Most SPA deal with systems assuming exponential or phase type distributions. Compact state-space representations (using binary decision diagrams [39] and efficient evaluation procedures are important topics of current research. Stochastic Model Checking, i.e. the systematic validation of SPA system model properties using temporal logic formulations is also on its way [34]. There is furthermore a clear trend to investigate systems with general distribution functions; when these investigations are successful, the dream of systematic hierarchical modeling with exact interface representation is reachable.

7 Conclusions

Performance evaluation has a long tradition. There are many success stories. However, since systems consist of both, hardware and software, performance is the major cause for project failure.

We surveyed in this contribution the standard elements and techniques of classical PE-methods. We showed their merits and the state-of-the-art. But we advocate more formal methods for PE since systems are getting more complex and more sophisticated in structure, operation and use.

Although the state-of-the-art is quite mature for integrated approaches, there are still many challenging problems;

- The Largeness Problem:
  - Research on Compositional Modeling and Analysis has to be extended.
  - Hierarchical Modeling with Exact Interfaces, not only at state-probability level, is an important goal.
  - Symmetry Exploitation has to be done efficiently.

- General Traffic Models:
  - Non-Markovian Distributions are required in many situations.
  - Discrete-Time Processes do occur e.g. in ATM-networks.
  - Long-Range Dependency of Processes appears in the Internet.
- QoS-Guarantees:
  - Functional, Temporal an Mixed Properties have to be determined.
  - Equivalences and Preorders for QoS-measures should be considered in combining system components.
  - Efficient bounding techniques are not yet available.
  - Verification and Model Checking is today possible only for small systems.
- Advancement in Design-Productivity:
  - Strict Formalization for Automation is necessary to improve the design cycle.
  - Development of complete Design-Tool-Chains is vital in modern industries.
- User Friendliness:
  - Design of Graphical and Textual Languages with the Same Meaning, and
  - Visualization of System Dynamics and QoS-Values are necessary preconditions for the acceptance by design engineers.

This list is by no means complete. But it clearly shows the variety of open problem classes offering exciting research topics for the future. All kind of innovative ideas are necessary to approach the ideal, a methodology which is theoretically well founded, automizable and accepted by the system designers.

Acknowledgment

I am grateful to the organizers of the summer school convincing me to summarize the evolution of PE methodology.Considering the variety of topics and working on their compact representation I got many impulses, could deepen my insight into this fascinating area.

I am also obliged to L. Kerber and M. Siegle for carefully reading the manuscript and making very valuable suggestions. Last not least, I am indebted to Ms. Hladky for her perseverance and obligingness in preparing the script concurrently to many other duties at our chair. I did not know how painful the use of todays commercial editors is. Again, Mr. Kerber gave us continuous support, Ms. Moog and others from our team spent many hours in supporting us.

References

1. M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems.* MIT Press, 1986.
2. A. Aldini, M. Bernardo, and R. Gorrieri. An Algebraic Model for Evaluating the Performance of an ATM-Switch with Explicit Rate Marking. In *Proceedings of the PAPM-Workshop*, 1999.

3. G. Balbo. Introduction to Stochastic Petri Nets. This volume.
4. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN System*, 14, 1987.
5. M. Bernardo, W.R. Cleaveland, S.T. Sims, and W.J. Stewart. TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems. In *FORTE/PSTV*, 1998.
6. G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queuing Networks and Markov Chains*. John Wiley&Sons, 1998.
7. E. Brinksma and H. Hermanns. Process Algebra and Markov Chains. This volume.
8. F. Bause and P. Kritzinger. *Stochastic Petri Nets*. Vieweg, Braunschweig, 1996.
9. E. Brinksma, J.P. Katoen, D. Latella, and R. Langerak. Partial–order models for quantitative extensions of LOTOS. *Computer Networks and ISDN Systems*, 30(9/10):925–950, 1998.
10. P. Buchholz. Markovian Process Algebra: Composition and Equivalence. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 11–30, Regensberg/Erlangen, July 1994. Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg.
11. J. P. Buzen. *Queuing network models of multiprogramming*. PhD thesis, Div. Eng. and Applied Physics, Harvad Univ., Cambridge, Mass., May 1971.
12. C. Cassandras. Discrete Event Systems: Modeling and Performance Analysis. Irwin Inc. and Aksen Associates, Inc., 1993.
13. G. Ciardo. Distributed and Structured Analysis Approaches to Study Large and Complex Systems. This volume, 2001.
14. L. Cohen. *The Single Server Queue*. North Holland, Amsterdam, 1969. revised 1982.
15. P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. IPA Dissertation Series 1999-10, CTIT Phd-Thesis Series 99-25, University of Twente, November 1999.
16. Erlang. The Application of the Theory of Probabilities in Telephone Administration. *Scandinavian H.C. Örsted Congress, Copenhagen*, 1920.
17. D. Ferrari. Considerations on the Insularity of Performance Evaluation. *IEEE Transactions on Software Engineering*, SE–12(6):678–683, June 1986.
18. H. J. Fromm, U. Hercksen, U. Herzog, K.-H. John, R. Klar, and W. Kleinöder. Experience with performance meassurement and modelling of a processor array. *IEEE Trans. on Computers*, C-32(1):15–31, January 1983.
19. F. Fischer, T. Hopfner, T. Kolloch, A. Muth, S. Petters, G. Färber, M. Dörfel, W. Dulz, R. Hofmann, A. Mitschele-Thiel, R. Münzenberger, F. Slomka, and U. Herzog. Rapid Prototyping von Realzeitsystemen. *it+ti*, 42(2):45–53, 2000.
20. D. Ferrari, G. Serazzi, and A. Zeigner. *Measurement and Tuning of Computer Systems*. Prentice Hall, Inc., Englewood Cliffs, 1983.
21. R. German. Non-Markovian Analysis. This volume.
22. D. Gross and C.M. Harris. *Fundamentals of Queuing Theory*. John Wiley&Sons, Second Edition, 1985.
23. N. Götz, U. Herzog, and M. Rettelbach. TIPP — Einführung in die Leistungsbewertung von verteilten Systemen mit Hilfe von Prozeßalgebren. In *Verteilte Systeme — Grundlagen und zukünftige Entwicklungen aus der Sicht des Sonderforschungsbereichs 182 "Multiprozessor- und Netzwerkkonfigurationen"*, pages 509–531. BI-Wissenschaftsverlag, 1993.
24. S. Gilmore. The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In G. Kotsis G. Haring, editor, *7th Internat.*

Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Wien, 1994.
25. R. Gorrieri. Stochastic Process Algebras; Past and Future. In J. Hillston and M. Silva, editors, *Proceedings of the PAPM'99-workshop*, pages 1–2, Prensas Universitarias de Zaragoza, 1999. Invited Talk at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications.
26. B. Haverkort. Markovian Models for Performance and Dependability Evaluation. This volume.
27. U. Herzog. Modellierung des Ablaufgeschehens in Networks - Modelling of Network Dynamics. In *Lectures and Tutorials, Information und Kommunikation*, number 8, pages 175–208. Oldenbourg, Munich, 1979.
28. U. Herzog. A Concept for Graph-Based Stochastic Process Algebras, Generally Distributed Activity Times and Hierarchical Modelling. Technical report IMMD VII 4/96, Universität Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen, May 1996.
29. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, 39(1-4):5–35, January 2000.
30. H. Hermanns, U. Herzog, and J.-P. Katoen. Process Algebra for Performance Evaluation. *Journal of Theoretical Computer Science*, Elsevier, to appear 2001.
31. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras - between LOTOS and Markov Chains. *Computer Networks and ISDN Systems*, 1998.
32. J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.
33. J. Hillston. Exploiting Structure in Solution: Decomposing Compositional Models. This volume.
34. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *2nd Int. Conference on Integrated Formal Methods*, pages 420–439, Dagstuhl, November 2000. Springer, LNCS 1945.
35. F.S. Hillier and G.J. Liebermann. *Introduction to Operations Research*. Holden Day Inc., San Francisco, 1967.
36. G. Haring, C. Lindemann, and M. Reiser. *Performance Evaluation: Origin and Directions*. Number 1769 in LNCS State-of-the-Art Survey. Springer, 2000.
37. J. Hillston and V. Mertsiotakis. A Simple Time Scale Decomposition Technique for Stochastic Process Algebras. *The Computer Journal*, 38(7):566–577, December 1995. Special issue: Proc. of the 3rd Workshop on Process Algebras and Performance Modelling.
38. H. Hermanns, V. Mertsiotakis, and M. Rettelbach. A Construction and Analysis Tool Based on the Stochastic Process Algebra TIPP. In *Proc. of 2nd Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 427–430. Springer, LNCS 1055, 1996.
39. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains. In B. Plateau, W.J. Stewart, and M. Silva, editors, *3rd Int. Workshop on the Numerical Solution of Markov Chains*, pages 188–207. Prensas Universitarias de Zaragoza, 1999.
40. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985.
41. U. Herzog and J. Rolia. Performance Validation Tools for Software/Hardware Systems. *Performance Evaluation*, 40, to appear in 2001.

42. International Teletraffic Congress, since 1955. Conference and proceedings every three years.
43. Raj Jain. *The Art of Computer Systems Performance Analysis.* J. Wiley, New York, 1991. I: Overview, II: Measurement, III: Probability, IV:Experimental Design, V: Simulation, VI: Queuing Models.
44. J.-P. Katoen and P.R. D'Argenio. General Distributions in Process Algebra. This volume.
45. R. Klar, P. Dauphin, F. Hartleb, R. Hofmann, B. Mohr, A. Quick, and M. Siegle. *Messung und Modellierung paralleler und verteilter Rechensystem.* Teubner, 1995.
46. L. Kleinrock. *Queueing Systems*, volume 1: Theory. John Wiley & Sons, 1975.
47. W. Kleinöder. *Stochastische Bewertung von Aufgabenstrukturen für Hierarchische Mehrrechnersysteme.* IMMD, University of Erlangen, August 1982. Dissertation am Lehrstuhl 7 des IMMD.
48. H. Kobayashi. *Modeling and Analysis — An Introduction to System Performance Evaluation Methodology.* Addison–Wesley, 1978.
49. F. Lemmen. *Spezifikationsgesteuertes Monitoring zur Integration der Leistungsbewertung in den formalen Entwurf von Kommunikationssystemen.* IMMD, University of Erlangen, 2000. Dissertation am Lehrstuhl 7 des IMMD.
50. K. Marzbani. Hierarchische Beschreibung und Analyse von Kommunikationssystemen mittels graphbasierter Prozessalgebren. Diplomarbeit, IMMD7, Universität Erlangen-Nürnberg, März 1997.
51. R. Gorrieri M. Bernardo, L. Donatiello. MPA, a Stochastic Process Algebra. Technical report, University of Bologna, 1994. Techn. Report to 94-10.
52. R. Milner. *Communication and Concurrency.* Prentice Hall, London, 1989.
53. M.K. Molloy. *On the integration of delay and throughput measures in distributed processing models.* PhD thesis, University of California, Los Angeles, 1981.
54. M.K. Molloy. *Fundamentals of Performance Modeling.* Macmillan-Colliev Macmillan, 1989.
55. V. Mertsiotakis and M. Silva. Throughput Approximation of Decision Free Processes Using Decomposition. In *Proc. of the 7th Int. Workshop on Petri Nets and Performance Models*, pages 174–182, St. Malo, June 1997. IEEE CS-Press.
56. C.A. Petri. Kommunikation mit Automaten, 1962. Dissertation an der Universität Bonn, Schriften des Instituts für Instrumentelle Mathematik Nr. 3.
57. C. Priami. Stochastic π-calculus with general distributions. In *Proc. of the 4th Workshop on Process Algebras and Performance Modelling*, pages 41–57, Torino, July 1996. Dpto. di Informatica, Universita di Torino. To appear.
58. J.A. Rolia and K.C. Sevcik. The Methods of Layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, August 1995.
59. T.L. Saaty. *Elements of Queuing Theory. With Applications.* McGraw-Hill, 1961.
60. W. Sanders. *UltraSAN User's Manual, Version 3.0.* University of Illinois, Urbana-Champaign, 1995.
61. W. Sanders and J.F. Meyer. Stochastic Activity Networks: Formal Definitions and Concepts. This volume.
62. A.L. Scherr. *An Analysis of Time-Shared Computer Systems.* Number 36 in Research Monograph. MIT Press, 1967.
63. F. Slomka, M. Dörfel, R. Münzenberger, and R. Hofmann. Hardware/Software Codesign and Rapid Prototyping of Embedded Systems. *IEEE Design and Test of Computers*, pages 28–38, 2000.
64. W.J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

65. W.J. Stewart. Numerical Analysis Methods. In G. Haring, Ch. Lindemann, and M. Reiser, editors, *Performance Evaluation: Origins and Directions*, LNCS 1769, pages 355–376. Springer, Berlin, Heidelberg, 2000.
66. R. v. Stieglitz. Messung und Bewertung des dynamischen Verhaltens eines Kommunikations–Prototypen für Breitband-ISDN (BERKOM). Dimplomarbeit, IMMD7, Universität Erlangen-Nürnberg, 1990.
67. R. Syski. *Introduction to Congestion Theory in Telephone Systems.* Oliver and Boyd, reprinted 1985.
68. P. Tran-Gia. *Analytische Leistungsbewertung verteilter Systeme.* Springer, 1996. in German.
69. Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, 1982.
70. I. Weigel. *Modelle für Entwurf und Bewertung eines dynamischen Lastverbunds lose gekoppelter Rechensysteme.* IMMD, Martensstraße 3, 91058 Erlangen, März 1994. Dissertation am Lehrstuhl für Rechnerarchitektur und -Verkehrstheorie, Universität Erlangen-Nürnberg.
71. P. Wimmer. Lastcharakterisierung in der automatisierten Fertigung am Beispiel der Leiterplattenbestückung. Diplomarbeit, IMMD7, Universität Erlangen-Nürnberg, Januar 1991.
72. C.M. Woodside, J.E. Neilson, D.C. Petriu, and S. Majumdar. The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. *IEEE Transactions on Computers*, 44(1):20–34, January 1995.

Markovian Models for Performance and Dependability Evaluation

Boudewijn R. Haverkort

Laboratory for Performance Evaluation and Distributed Systems
Department of Computer Science, RWTH Aachen, 52056 Aachen, Germany
haverkort@cs.rwth-aachen.de

Abstract. Markovian models have been used for about a century now for the evaluation of the performance and dependability of computer and communication systems. In this paper, we give a concise overview of the most widely used classes of Markovian models, their solution and application.
After a brief introduction to performance and dependability evaluation in general, we introduce discrete-time Markov chains, continuous-time Markov chains and semi-Markov chains. Stepwisely, we develop the main equations that govern the steady-state and the transient behaviour of such Markov chains. We thereby emphasise on intuitively appealing explanations rather than on mathematical rigor. The relation between the various Markov chain types is explained in detail. Then, we discuss means to numerically solve the systems of linear equations (both direct and iterative ones) and the systems of differential equations that arise when solving for the steady-state and transient behaviour of Markovian models.

1 Introduction

Markovian models have inherited their name form the pioneering work by the Russian mathematician A.A. Markov around the turn of the twentieth century (see Figure 1). He introduced finite-state Markov chains in [49]; a translation of another important article of his hand appears in Appendix B of [37]. In fact, his work launched the area of stochastic processes. In the first two decades of the twentieth century, the Danish mathematician A.K. Erlang (see Figure 2) applied Markovian techniques (then not yet named as such) to solve capacity planning problems for the Copenhagen Telephone Company [20]. His models were soon adapted by others, among others by the Britisch Post Office; one of his first models will be presented later in this paper. The Russian mathematician A.N. Kolmogorov (see Figure 3) developed the theory for Markov chains with infinite (denumerable and continuous) state spaces in the 1930's [46].

Throughout the twentieth century the work of these pioneers became better understood and more wide-spread. These days, Markov chains and stochastic processes form the basis for model-based system evaluations in many areas of science and engineering. For instance, in biology to model growth and decay of

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 38–83, 2001.

Fig. 1. Andrei Andreevich Markov (* June 14, 1856; † July 20, 1922)

Fig. 2. Agner Krarup Erlang (* January 1, 1878; † February 3, 1929)

Fig. 3. Andrey Nikolaevich Kolmogorov (* April 25, 1903; † October 20, 1987)

populations, in physics to model interactions between elementary particles, in chemical engineering to model (chain) reactions between molecules or to model mixing processes, in management science to model the flow of commodities in logistic or flexible manufacturing systems or to model the availability of production lines and, most notably, in computer and communication science and engineering to model system performance and dependability in a wide variety of settings. In this paper, we focus on *the use of Markovian models for the performance and dependability evaluation of computer and communication systems.*

But let us first take one step back, and address the question what performance or dependability evaluation really is. Performance or dependability evaluation is the craft that tries to answer questions related to the performance or dependability of systems. Typical questions take the following form:

(i) how many clients can this server adequately support?
(ii) what is the typical response time to load a WWW page from MIT?
(iii) how large should the buffer space in this IP router be to guarantee a packet loss ratio of less than 10^{-6}?
(iv) how many jobs can be processed before a system failure occurs?
(vi) how long does it take before this system crashes?

The above questions can only be answered when they are made more exact, i.e., when some of the informally stated requirements or aims are made concrete. For instance, we will have to define what "adequately" really means in question (i), or what "typical" means in question (ii). More precisely, we have to define

clearly what the *measure of interest* is, in order to express the performance or dependability criterion we are interested in, in the best possible way. Before we will come to this issue, we will restrict the class of evaluation techniques considerably. In this paper, we will only address so-called *model-based* evaluation techniques, meaning that we are not addressing measurement-based techniques such as benchmarking. Even though measurement-based evaluation techniques are very important and accurate (they address the real system, or at least a prototype) these methods are also very costly, and sometimes even inappropriate, for instance when the interest is in very rare events; a measurement-based approach then takes too long to be practically feasible. Hence, we restrict ourselves to model-based evaluation techniques, meaning that we have to develop models, in order to evaluate the system under study. According to [36], a *model* is a "*small-scale reproduction or representation of something*" and *modelling* is "*the* art *of making models*". The latter definition states an important aspect of model-based performance and dependability evaluation: the construction of appropriate models is a challenging task for which, as of yet, no general recipe is available.

Let us now come back to the measures of interest in a performance or dependability evaluation. The choice of measure strongly depends on the standpoint of the model user (the person using the results of the model evaluation). System engineers are most often interested in *system-oriented measures* like queue lengths, component utilisations or the number of operational components. For system users, seeing the system as a service-providing black box, these measures are not that interesting; for them what counts is how fast or how many service invocations can be performed per unit of time. Examples of such *user-oriented measures* are waiting times, throughput and downtime minutes per year. On top of this classification comes the question how detailed the measure of interest should be evaluated: does a mean value suffice, or is knowledge of higher moments or even of the complete distribution necessary? Furthermore, is the measure to be evaluated for a particular time instance in the operation of the system, that is, is there an interest in so-called *transient measures*, or is the interest more in long-term average values, that is, in so-called *steady-state measures*.

With the class of Markovian models we will address in the rest of this paper, we have available a versatile modelling formalism, allowing us (i) to model system performance and dependability at various levels of detail, (ii) to study a wide variety of user- and system-oriented measures, at (iii) either in steady-state or at some time instance t.

It should be noted that by the availability of good software tools for performance and dependability evaluation, the actual Markov chains being used and solved often remain hidden from the end-user. That is, users specify their performance or dependability model using some high-level modelling language, for instance based on queueing networks, stochastic Petri nets or stochastic process algebras of some sort, from which the underlying Markov chain can automatically be generated and analysed. The analysis results are then presented such that they can be interpreted correctly in the context of the high-level model,

so that, in fact, the translations to and from the Markov chain level remain transparant. High-level specification techniques for Markov chains are an active area of research, but are not addressed in more detail in this paper; the interested reader is referred to a number of surveys addressing these issues [31,30,29] as well as to three other papers in this volume [6,10,60].

A final point of notice is the following. Model-based performance and dependability evaluation necessarily have to be based on abstractions of the real system. In that sense, it is intrinsically approximate. This is both a strength, as it can be applied always (albeit more or less accurate), and a weakness, as its accuracy is not known in advance. In any case, the results from an evaluation should be interpreted with care; the results are never more accurate than the numerical parameters used in the models! Furthermore, note that even though model-based performance and dependability evaluation provides us with numbers, the insight gained in the (functional) operation of the system under study is often even more important. As Alan Scherr, IBM's time-sharing pioneer, puts it in an interview with *Communications of the ACM* [22]: "model-based performance evaluation is about finding those 10% of the system that explains 90% of its behaviour".

After this more general introduction, the rest of this paper completely focusses on Markov chains. In Section 2 we introduce discrete-time Markov chains, followed by the introduction of semi-Markov chains and continuous-time Markov chains in Section 3 and Section 4, respectively. We then address techniques to solve these Markov chains with respect to their steady-state and their transient-state probabilities in Section 5 and Section 6, respectively. A variety of important issues not addressed in this paper is presented in Section 7. The paper ends with Section 8.

2 Discrete-Time Markov Chains

We define discrete-time Markov chains in Section 2.1, followed by a derivation of the steady-state and transient state probabilities in Section 2.2. We comment on the state residence time distribution in Section 2.3 and discuss convergence properties in Section 2.4.

2.1 Definition

Discrete-time Markov chains (DTMCs) are a class of stochastic processes. A stochastic process can be regarded as a family of random variables $\{X_t, t \in \mathcal{T}\}$, of which each instance X_t is characterisedby a distribution function. The index set $\mathcal{T}$ is mostly associated with the passing of time. In DTMCs, time passes in discrete steps, so that the subsequent time instances are denumerable and can be seen as elements of $\mathbb{N}$, hence, one typically denotes a DTMC as $\{X_n, n \in \mathbb{N}\}$. The continuous counterpart to DTMCs, that is, continuous-time Markov chains, will be discussed in Section 4.

The set of values the random variables X_n can assume is denoted the *state space* $\mathcal{I}$ of the DTMC. In performance and dependability evaluation, most often the state space of a DTMC is denumerable. We do restrict ourselves to that case here. Given a denumerable set $\mathcal{I}$, it can either be finite or infinitely large. We will only address the finite case here; a few remarks with respect to denumerable infinite state space will be given in the examples in Section 4.

The fact that we deal with a finite-state discrete-time stochastic process does not directly imply that we are dealing with a DTMC. The distinctive property of a DTMC is that the *Markov property* has to hold for it. This means that given the current state of the DTMC, the future evolution of the DTMC is totally described by the current state, and is independent of past states. This property is intuitively so appealing, that one sometimes tends to forget that it is a very special property. Mathematically, the Markov property can be described as follows. Assuming $\mathcal{T} = \{0, 1, 2, \cdots\}$ and $\mathcal{I} = \{i_0, i_1, \cdots\}$ we have

$$\Pr\{X_{n+1} = i_{n+1} | X_0 = i_0, \cdots, X_n = i_n\} = \Pr\{X_{n+1} = i_{n+1} | X_n = i_n\}.$$

From this equation we see that the future (at time instance $n+1$) only depends on the current state (at time instance n) and is independent of states assumed in the past (time instances 0 through $n-1$). At this point, we also note that the DTMCs we study are *time-homogeneous*, which means that the actual time instances are not important, only their relative differences, that is:

$$\Pr\{X_{n+1} = i | X_n = j\} = \Pr\{X_{m+1} = i | X_m = j\}, \quad \text{for all } n, m \in \mathbb{N},$$

with $i, j \in \mathcal{I}$. This means that in a time-homogeneous DTMC the state-transition behaviour itself does not change over time.

We now define the conditional probability $p_{j,k}(m, n) = \Pr\{X_n = k | X_m = j\}$, for all $m = 0, \cdots, n$, i.e., the probability of going from state $j \in \mathcal{I}$ at time $m \in \mathbb{N}$ to state $k \in \mathcal{I}$ at time $n \in \mathbb{N}$. Since we deal with time-homogeneous Markov chains, these *transition probabilities* only depend on the time difference $l = n - m$. We can therefore denote them as $p_{j,k}(l) = \Pr\{X_{m+l} = k | X_m = j\}$, the so-called *$l$-step transition probabilities*. The *1-step transition probabilities* are simple denoted $p_{j,k}$ (the parameter 1 is omitted). The 0-step probabilities are defined as $p_{j,k}(0) = 1$, whenever $j = k$, and 0 elsewhere. The initial distribution $\underline{\pi}(0)$ of the DTMC is defined as $\underline{\pi}(0) = (\pi_0(0), \cdots, \pi_{|\mathcal{I}|}(0))$. By iteratively applying the rule for conditional probabilities, it can easily be seen that

$$\Pr\{X_0 = i_0, X_1 = i_1, \cdots, X_n = i_n\} = \pi_{i_0}(0) \cdot p_{i_0,i_1} \cdots p_{i_{n-1},i_n}. \tag{1}$$

This implies that the DTMC is totally described by the initial probabilities and the 1-step probabilities $p_{i,j}$. The 1-step probabilities are summarised in the *state-transition probability matrix* $\mathbf{P} = (p_{i,j})$. The matrix $\mathbf{P}$ is a *stochastic matrix* because all its entries $p_{i,j}$ satisfy $0 \leq p_{i,j} \leq 1$, and $\sum_j p_{i,j} = 1$, for all i.

A DTMC can be visualised as a labeled directed graph with the elements of $\mathcal{I}$ as vertices. A directed edge with label $p_{i,j}$ exists between vertices i and j whenever $p_{i,j} > 0$. Such representations of Markov chains are called *state*

transition diagrams. Notice the similarity with the usual graphical notation for finite-state machines (FSMs). In fact, a DTMC can be viewed as an FSM in which the successor function is specified in a probabilistic manner, that is, given state i, the next state will be state j with probability $p_{i,j}$.

Example 1. Graphical representation of a DTMC. In Figure 4 we show the state transition diagram for the DTMC with state-transition probability matrix

$$\mathbf{P} = \frac{1}{10}\begin{pmatrix} 6 & 2 & 2 \\ 1 & 8 & 1 \\ 6 & 0 & 4 \end{pmatrix}. \tag{2}$$

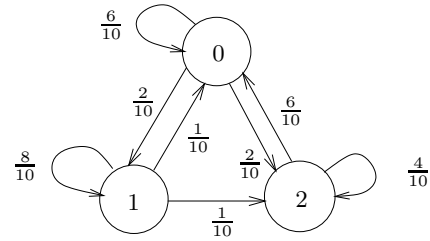


Fig. 4. State transition diagram for the example DTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

2.2 Transient and Steady-State Probabilities

We can now proceed to calculate the 2-step probabilities of a DTMC with state-transition probability matrix $\mathbf{P}$. We have

$$p_{i,j}(2) = \Pr\{X_2 = j | X_0 = i\} = \sum_{k \in \mathcal{I}} \Pr\{X_2 = j, X_1 = k | X_0 = i\}, \tag{3}$$

since in going from state i to state j in two steps, any state $k \in \mathcal{I}$ can be visited as intermediate state. Now, due to the rule of conditional probability as well as the Markov property, we can write

$$\begin{aligned} p_{i,j}(2) &= \sum_{k \in \mathcal{I}} \Pr\{X_2 = j, X_1 = k | X_0 = i\} \\ &= \sum_{k \in \mathcal{I}} \Pr\{X_1 = k | X_0 = i\} \Pr\{X_2 = j | X_1 = k, X_0 = i\} \\ &= \sum_{k \in \mathcal{I}} \Pr\{X_1 = k | X_0 = i\} \Pr\{X_2 = j | X_1 = k\} \\ &= \sum_{k \in \mathcal{I}} p_{i,k} p_{k,j}. \end{aligned} \tag{4}$$

In the last summation we recognise the matrix product. Thus, the 2-step probabilities $p_{i,j}(2)$ are elements of the matrix $\mathbf{P}^2$. The above derivation can be applied iteratively, yielding the n-step probabilities $p_{i,j}(n)$ as elements of the matrix $\mathbf{P}^n$. For the 0-step probabilities we find the matrix $\mathbf{I} = \mathbf{P}^0$. The equation that establishes a relation between the $(m+n)$-step probabilities and the m- and n-step probabilities, that is,

$$\mathbf{P}^{m+n} = \mathbf{P}^m \mathbf{P}^n, \tag{5}$$

is known as the *Chapman-Kolmogorov equation*.

The probability of residence in state j after n steps, that is $\pi_j(n)$, can be obtained by conditioning:

$$\begin{aligned} \pi_j(n) = \Pr\{X_n = j\} &= \sum_{i \in \mathcal{I}} \Pr\{X_0 = i\} \Pr\{X_n = j | X_0 = i\} \\ &= \sum_{i \in \mathcal{I}} \pi_i(0) p_{i,j}(n). \end{aligned} \tag{6}$$

Writing this in matrix-vector notation, with $\underline{\pi}(n) = (\pi_0(n), \pi_1(n), \cdots)$, we arrive at

$$\underline{\pi}(n) = \underline{\pi}(0)\mathbf{P}^n. \tag{7}$$

Since the index n in (7) can be interpreted as the step-count in the DTMC, this equation expresses the time-dependent or transient behaviour of the DTMC.

Example 2. Transient behaviour of a DTMC. Let us compute $\underline{\pi}(n) = \underline{\pi}(0)\mathbf{P}^n$ for $n = 1, 2, 3, \cdots$ with $\mathbf{P}$ as given in (2), and $\underline{\pi}(0) = (1, 0, 0)$. Clearly, $\underline{\pi}(1) = \underline{\pi}(0)\mathbf{P} = (0.6, 0.2, 0.2)$. Then, $\underline{\pi}(2) = \underline{\pi}(0)\mathbf{P}^2 = \underline{\pi}(1)\mathbf{P} = (0.50, 0.28, 0.22)$. We proceed with $\underline{\pi}(3) = \underline{\pi}(2)\mathbf{P} = (0.460, 0.324, 0.216)$. We observe that the successive values for $\underline{\pi}(n)$ seem to converge somehow, and that the elements of all the vectors $\underline{\pi}(n)$ always sum to 1.

For many DTMCs (but not all; we will discuss conditions in Section 2.4) all the rows in $\mathbf{P}^n$ converge to a common limit when $n \to \infty$. For the time being, we assume that such a limit indeed exists. Defining $\underline{v} = (\cdots, v_j, \cdots)$ as

$$v_j = \lim_{n \to \infty} \pi_j(n) = \lim_{n \to \infty} \Pr\{X_n = j\} = \lim_{n \to \infty} \sum_{i \in \mathcal{I}} \pi_i(0) p_{i,j}(n). \tag{8}$$

Writing this in matrix-vector notation, we obtain

$$\underline{v} = \lim_{n \to \infty} \underline{\pi}(n) = \lim_{n \to \infty} \underline{\pi}(0)\mathbf{P}^n. \tag{9}$$

However, we also have

$$\underline{v} = \lim_{n \to \infty} \underline{\pi}(n+1) = \lim_{n \to \infty} \underline{\pi}(0)\mathbf{P}^{n+1} = \left(\lim_{n \to \infty} \underline{\pi}(0)\mathbf{P}^n \right) \mathbf{P} = \underline{v}\mathbf{P}. \tag{10}$$

Hence, whenever the limiting probabilities $\underline{v}$ exist, they can be obtained by solving the system of linear equations

$$\underline{v} = \underline{v}\mathbf{P} \quad \Rightarrow \quad \underline{v}(\mathbf{I} - \mathbf{P}) = \underline{0}, \tag{11}$$

with, since $\underline{v}$ is a probability vector, $\sum_i v_i = 1$, and $0 \leq v_i \leq 1$. The equivalent form on the right, i.e., $\underline{v}(\mathbf{I} - \mathbf{P}) = \underline{0}$, will be discussed in Section 4 in relation to CTMCs.

The vector $\underline{v}$ is called the *stationary* or *steady-state probability vector* of the DTMC, which, for the DTMCs we will encounter, will most often uniquely exist. Furthermore, in most of the practical cases we will encounter, this steady-state probability vector will be independent of the initial state probabilities $\underline{\pi}(0)$.

Example 3. Steady-state probability vector calculation. Let us compute $\underline{v} = \underline{v}\mathbf{P}$ with $\mathbf{P}$ as in (2) and compare it to the partially converged result obtained there. Denoting $\underline{v} = (v_0, v_1, v_2)$ we derive from the system of three linear equations that $v_0 = v_1$ and $v_2 = v_0/2$. Using the fact that $v_0 + v_1 + v_2 = 1$ (normalisation) then gives us $\underline{v} = (\frac{4}{10}, \frac{4}{10}, \frac{2}{10})$.

The steady-state probabilities can be interpreted in two ways. One way is to see them as the long-run proportion of time the DTMC "spends" in the respective states. The other way is to regard them as the probabilities that the DTMC would be in a particular state if one would take a snapshot after a very long time. It is important to note that for large values of n state changes do still take place!

2.3 State-Residence Time Distribution

The matrix $\mathbf{P}$ describes the 1-step state transition probabilities. If, at some time instance n, the state of the DTMC is i, then, at time instance $n+1$, the state will still be i with probability $p_{i,i}$, and will be $j \neq i$ with probability $1 - p_{i,i} = \sum_{j \neq i} p_{i,j}$. For time instance $n+1$, a similar reasoning holds, so that the probability of still residing in state i at time instance $n+2$ (given residence there at time instance n and $n+1$) equals $p_{i,i}^2$. Taking this further, the probability to reside in state i for exactly m consecutive epochs equals $(1 - p_{i,i})p_{i,i}^{m-1}$, that is, there are $m-1$ steps in which the possibility (staying in i) with probability $p_{i,i}$ is taken, and one final step with probability $1 - p_{i,i}$ where indeed a step towards another state $j \neq i$ is taken. Interpreting leaving state i as a success and staying in state i as a failure, i.e., a failure to leave, we see that the state residence times in a DTMC obey a *geometric distribution*. The expected number of epochs in state i then equals $1/(1 - p_{i,i})$ and the variance of the number of epochs in state i then equals $p_{i,i}/(1 - p_{i,i})^2$.

The fact that the state residence times in a DTMC are geometrical distributions need not be a surprise. From the Markov property, we know that only the actual state, at any time instance, is of importance in determining the future, irrespective of the residence time in that state. The geometric distribution is the

only discrete distribution exhibiting this *memoryless property*, that is, when a random variable M is geometrically distributed the following holds:

$$\Pr\{M = n + m | M > n\} = \Pr\{M = m\}, \quad m \geq 1.$$

2.4 Convergence Properties

Previously, we stated that the steady-state probability distribution of a DTMCs can be determined when the DTMC fulfills certain conditions. In this section we discuss concisely a number of properties of DTMCs that help us in deciding whether a DTMC has a unique steady-state probability distribution or not.

Let us start with a classification of the states in a DTMC. A state $j \in \mathcal{I}$ is said to be *reachable* from state $i \in \mathcal{I}$ if, for some value n, $p_{i,j}(n) > 0$, which means that there is a step number for which there is a nonzero probability of going from state i to j. For such a pair of states, we write $i \to j$. If $i \to j$ and $j \to i$, then i and j are said to be *communicating states*, denoted $i \sim j$. Clearly, the communicating relation ($\sim$) is (i) transitive: if $i \sim j$ and $j \sim k$ then $i \sim k$, (ii) symmetric: by its definition in terms of $\to$, $i \sim j$ is equivalent to $j \sim i$, and (iii) reflexive: for $n = 0$, we have $p_{i,i}(0) = 1$, so that $i \to i$ and therefore $i \sim i$. Consequently, $\sim$ is an *equivalence relation* which partitions the state space in communicating classes. If all the states of a DTMC belong to the same communicating class, the DTMC is said to be *irreducible*. If not, the DTMC is *reducible*.

The *period* $d_i \in \mathbb{N}$ of state i is defined as the greatest common divisor of those values n for which $p_{i,i}(n) > 0$. When $d_i = 1$, state i is said to be *aperiodic*, in which case, at every time step there is a non-zero probability of residing in state i. It has been proven that within a communicating class all states have the same period. Therefore, one can also speak of periodic and aperiodic communicating classes, or, in case of an irreducible DTMC, of an *aperiodic* or *periodic DTMC*.

A state i is said to be *absorbing* when $\lim_{n\to\infty} p_{i,i}(n) = 1$. Recall that for an absorbing state i we have $\sum_{j\neq i} p_{i,j} = 0$. When there is only a single absorbing state, the DTMC will, with certainty, reach that state for some value of n.

A state is said to be *transient* or *non-recurrent* if there is a nonzero probability that that state is not visited again at some point in the future. If this is not the case, the state is said to be recurrent. For *recurrent* states, we can address the time between successive visits. Let $f_{i,j}(n)$ denote the probability that exactly n steps after leaving state i, state j is visited for the first time. Consequently, $f_{i,i}(n)$ is the probability that exactly n steps are taken between two successive visits to state i. Defining $f_{i,i} = \sum_n f_{i,i}(n)$, it follows that if $f_{i,i} = 1$, then state i is recurrent. If state i is nonrecurrent then $f_{i,i} < 1$. In the case $f_{i,i} = 1$ we can make a further classification based upon the mean recurrence time m_i of state i, defined as $m_i = \sum_{n=1}^{\infty} n f_{i,i}(n)$. A recurrent state i is said to be *positive recurrent* (or recurrent non-null) if the mean recurrence time m_i is finite. If m_i is infinite, state i is said to be *null recurrent*.

Having defined the above properties, the following theorem expresses when a DTMC has a (unique) steady-state probability distribution.

Theorem 1. Steady-state probability distributions in a DTMC. *In an irreducible and aperiodic DTMC with positive recurrent states:*

- *for all j, the limiting probability $v_j = \lim_{n\to\infty} \pi_j(n) = \lim_{n\to\infty} p_{i,j}(n)$ does exist;*
- *$\underline{v}$ is independent of the initial probability distribution $\underline{\pi}(0)$;*
- *$\underline{v}$ is the unique steady-state probability distribution.*

In typical performance and dependability models, the Markov chains will be irreducible and aperiodic. When dealing with continuous-time Markov chains, similar conditions apply as for DTMC.

3 Semi-Markov Chains

We define semi-Markov chains in Section 3.1 and give an alternative interpretation of their dynamics in Section 3.2.

3.1 SMCs as Generalisation of DTMCs

We can leave the discrete-time domain and move to the continuous-time domain by associating with every state in a DTMC a positive residence time distribution $F_i(t)$ and density $f_i(t)$. In doing so, we end up with a *semi-Markov chain* (SMC), which is fully described by the matrix with 1-step probabilities $\mathbf{P}$ (as known from DTMCs), the initial probability vector $\underline{\pi}(0)$ and the vector of state residence distributions $\underline{F}(t) = (F_1(t), \cdots, F_{|\mathcal{I}|}(t))$. A simple interpretation of an SMC is the following. At epochs when the state changes take place (transition epochs), the SMC behaves as a DTMC in the sense that the state changes are completely governed by the state transition probability matrix $\mathbf{P}$, and are independent of the past. When state i is entered, a random amount of time has to be passed, distributed according to $F_i(t)$, before the next state transition takes place.

To obtain the steady-state probabilities of an SMC, we first solve the steady-state probabilities for the so-called *embedded DTMCs* characterisedby $\mathbf{P}$. Since the SMC behaviour at transition epochs is the same as for this DTMC, we can compute the steady-state probabilities $\underline{v}$ for the embedded DTMC in the usual way. Now, we have to compute the average state residence times h_i for all states i in the SMC. We do this directly from the state residence time distributions:

$$h_i = \int_0^\infty t f_i(t) dt.$$

We then obtain the steady-state probabilities in the SMC by taking these residence times into account, as follows:

$$\pi_i = \frac{v_i h_i}{\sum_j v_j h_j}, \quad \text{for all } i.$$

Note that for the steady-state probabilities of the SMC only the mean state residence times h_i are of importance. Hence, in many applications, these mean

values are given directly, so that the explicit integration above does not need to be performed.

The computation of transient state probabilities for an SMC is far more complex than for DTMCs (and for CTMCs). A derivation of the relevant equations as well as their solution go beyond the scope of this paper, but can be found in [48].

3.2 Alternative View on SMCs

We can also view an SMCs in a slightly different, but equivalent, way. Consider a DTMC in which the transition probabilities are dependent on the time already spent in (current) state i since the last entrance there, but not on states visited before entering state i nor on any previous residence times. Thus, we deal with a time-dependent probability matrix $\mathbf{K}(t)$ known as the *kernel of the SMC*, where an entry $k_{i,j}(t)$ provides the probability that, after having entered state i, it takes at most t time units to switch to state j, given that no transition to any other state takes place.

From $\mathbf{K}(t)$, we can derive two well-known other quantities. First of all, the limit $p_{i,j} = \lim_{t\to\infty} k_{i,j}(t)$ expresses the probability that once state i has been entered, the next state will be j. The thus resulting probabilities indeed coincide with the entries of $\mathbf{P}$ for the embedded DTMC in Section 3.1. Furthermore, the state residence time distribution $F_i(t)$ can be written as $F_i(t) = \sum_j k_{i,j}(t)$.

Hence, once $\mathbf{K}(t)$ is known, both $\mathbf{P}$ and $\underline{F}(t)$ can be derived and the solution procedure of Section 3.1 can be applied.

4 Continuous-Time Markov Chains

In this section, we focus on continuous-time Markov chains (CTMCs). We first present how CTMCs can be seen as generalisations of DTMCs, by enhancing them with negative exponential state residence time distributions in Section 4.1. We then present the evaluation of the steady-state and transient behaviour of CTMCs in Section 4.2.

4.1 From DTMC to CTMC

In DTMCs time progresses in abstract steps. With CTMCs, as for SMCs, we associate positive state-residence time distributions with each state; hence we address Markov chains in continuous-time. In SMCs, we associated general residence time distributions with states. As a result, the state transition probability matrix $\mathbf{K}(t)$ became time dependent, so that the *complete state* of an SMC is given by the current state number i *and* the time already spent in that state (denoted in the sequel as t_{res}).

With CTMCs, we strive for a considerably more simple notion of state. We will chose the state residence time distribution such that the current state index i describes the state of the chain completely. This can only be achieved when

the chosen state residence time distribution is memoryless, so that it does not matter what the actual value of t_{res} is. In doing so, the Markov property is valid, and reads for the case at hand, for all non-negative $t_0 < t_1 < \cdots < t_{n+1}$ and $x_0, x_1, \cdots, x_{n+1}$:

$$\begin{array}{c} \Pr\{X(t_{n+1}) = x_{n+1} | X(t_0) = x_0, \cdots, X(t_n) = x_n\} \\ = \\ \Pr\{X(t_{n+1}) = x_{n+1} | X(t_n) = x_n\}, \end{array} \tag{12}$$

hence, the probability distribution for the $(n+1)$-th state residence time only depends on the current (n-th) state and neither on the time passed in the current state, nor on states visited previously.

The only memoryless continuous-time distribution is the exponential distribution. Thus, we have to associate with every state i in a CTMC a parameter μ_i describing the rate of an exponential distribution; the residence time distribution in state i then equals

$$F_i(t) = 1 - e^{-\mu_i t},\ t \geq 0. \tag{13}$$

The vector $\underline{\mu} = (\cdots, \mu_i, \cdots)$ thus describes the state residence time distributions in the CTMC. To be precise, this vector describes the rates of these distributions, but these rates fully characterise the distribution. We can still use the state transition probability matrix $\mathbf{P}$ to describe the state transition behaviour. The initial probabilities remain $\underline{\pi}(0)$. The dynamics of the CTMC can now be interpreted as follows. When state i is entered, this state will remain for a random amount of time, distributed according to the state residence distribution $F_i(t)$. After this delay, a state change to state j will take place with probability $p_{i,j}$. To ease understanding at this point, assume that $p_{i,i} = 0$ for all i; we come back to this issue below.

Instead of associating with every state just one negative exponentially distributed delay, it is also possible to associate as many delays with a state as there are transition possibilities. We therefore define the matrix $\mathbf{Q}$ with $q_{i,j} = \mu_i p_{i,j}$, in case $i \neq j$, and $q_{i,i} = -\sum_{j \neq i} q_{i,j} = -\mu_i$. Since $p_{i,i} = 0$, we have $q_{i,i} = -\mu_i$. Using this notation allows for the following interpretation. When entering state i, for those states j that can be reached from i, i.e., for those with $q_{i,j} > 0$, a random variable is thought to be drawn, according to the (negative exponential) distributions $F_{i \to j}(t) = 1 - e^{-q_{i,j} t}$. These distributions model the delay perceived in state i when going to j. One of the "drawn" delays will be the smallest, meaning that the transition corresponding to that delay will actually occur before any of the others (race condition: the faster one wins). This interpretation is correct due to the special properties of the negative exponential distribution. Let us first address the state residence times. In state i, the time it takes to reach state j is exponentially distributed with rate $q_{i,j}$. When there is more than one possible successor state, the next state will be such that the residence time in state i is minimised (race condition). However, the minimum value of a set of of exponentially distributed random variables with rates $q_{i,j}$ $(j \neq i)$ is again an exponentially distributed random variable, with as rate the sum $\sum_{j \neq i} q_{i,j}$ of the

original rates. This sum is, however, exactly equal to the rate μ_i of the residence time in state i.

A second point to verify is whether the state-transition behaviour is still the same. In general, if we have n negative exponentially distributed random variables X_k (with rates l_k), then X_i will be the minimum of them with probability $l_i / \sum_k l_k$. In our case, we have a number of competing delays when starting from state i, which are all negative exponentially distributed random variables (with rates $q_{i,j}$). The shortest one will then lead to state j with probability

$$\frac{q_{i,j}}{\sum_{k \neq i} q_{i,k}} = \frac{p_{i,j}\mu_i}{\mu_i} = p_{i,j}, \tag{14}$$

so that also the state-transition behaviour is as required.

Let us now discuss the case where $p_{i,i} > 0$, that is, the case where, after having resided in state i for an exponentially distributed period of time (with rate μ_i), there is a positive probability of staying in i for another period. In particular, we have seen in Section 2 that the state residence distributions in a DTMC obey a geometric distribution (measured in "visits"), with mean $1/(1-p_{i,i})$ for state i. Hence, if we decide that the expected state residence *time* in the CTMC constructed from the DTMC is $1/\mu_i$, the time spent in state i *per visit* should on average be $(1-p_{i,i})/\mu_i$. Hence, the rate of the negative exponential distribution associated with that state should equal $\mu_i/(1-p_{i,i})$. Using this rate in the above procedure, we find that we have to assign the following transition rates for $j \neq i$:

$$q_{i,j} = \frac{\mu_i p_{i,j}}{1-p_{i,i}} = \mu_i \frac{p_{i,j}}{1-p_{i,i}} = \mu_i \Pr\{\text{jump } i \to j | \text{leave } i\}, \quad j \neq i, \tag{15}$$

that is, we have renormalised the probabilities $p_{i,j}$ $(j \neq i)$ such that they make up a proper distribution. To conclude, if we want to associate a negative exponential residence time with rate μ_i to state i, we can do so by just normalising the probabilities $p_{i,j}$ $(j \neq i)$ appropriately.

4.2 Evaluating the Steady-State and Transient Behaviour

CTMCs can be depicted conveniently using state-transition diagrams. i.e., as labelled directed graphs, with the states of the CTMC represented by the vertices and an edge between vertices i and j $(i \neq j)$ whenever $q_{i,j} > 0$. The edges in the graph are labelled with the corresponding rates.

Formally, a CTMC can be described by an *(infinitesimal) generator matrix* $\mathbf{Q} = (q_{i,j})$ and initial state probability vector $\underline{\pi}(0)$. Denoting the system state at time $t \in \mathcal{T}$ as $X(t)$, we have, for $h \to 0$:

$$\Pr\{X(t+h) = j | X(t) = i\} = \Pr\{X(h) = j | X(0) = i\} = q_{i,j}h + \mathbf{o}(h), \quad i \neq j, \tag{16}$$

where $\mathbf{o}(h)$ is a term that goes to zero faster than h. This result follows because the CTMC is time-homogeneous and the fact that the state residence times are negative exponentially distributed; in fact, (16) represents the first-order

Taylor/MacLaurin series expansion of $1-e^{-q_{i,j}h}$ around 0. The value $q_{i,j}$ $(i \neq j)$ is the rate at which the current state i changes to state j. Denote with $\pi_i(t)$ the probability that the state at time t equals i: $\pi_i(t) = \Pr\{X(t) = i\}$. Given $\pi_i(t)$, we can compute the evolution of the Markov chain in the period $[t, t+h)$ as follows:

$$\pi_i(t+h) = \pi_i(t) \Pr\left\{\begin{array}{c}\text{do not depart from } i\\ \text{during } [t,t+h)\end{array}\right\} + \sum_{j\neq i} \pi_j(t) \Pr\left\{\begin{array}{c}\text{go from } j \text{ to } i\\ \text{during } [t,t+h)\end{array}\right\}$$

$$= \pi_i(t)\left(1 - \sum_{j\neq i} q_{i,j}h\right) + \left(\sum_{j\neq i} \pi_j(t) q_{j,i}\right) h + \mathbf{o}(h). \tag{17}$$

Now, using the earlier defined notation $q_{i,i} = -\sum_{j\neq i} q_{i,j}$, we have

$$\pi_i(t+h) = \pi_i(t) + \left(\sum_{j\in\mathcal{I}} \pi_j(t) q_{j,i}\right) h + \mathbf{o}(h). \tag{18}$$

Rearranging terms, dividing by h and taking the limit $h \to 0$, we obtain

$$\pi_i'(t) = \lim_{h\to 0} \frac{\pi_i(t+h) - \pi_i(t)}{h} = \sum_{j\in\mathcal{I}} q_{j,i}\pi_j(t), \tag{19}$$

which in matrix-vector notation has the following form:

$$\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}, \quad \text{given } \underline{\pi}(0). \tag{20}$$

We thus find that the time-dependent or *transient state probabilities* in a CTMC are described by a system of linear differential equations.

In many cases, however, the *transient behaviour* $\underline{\pi}(t)$ of the Markov chain is more than we really need. For performance evaluation purposes we are often already satisfied when we are able to compute the long-term or *steady-state probabilities* $\pi_i = \lim_{t\to\infty} \pi_i(t)$. When we assume that a steady-state distribution exists, this implies that the above limit exists, and thus that $\lim_{t\to\infty} \pi_i'(t) = 0$. Consequently, for obtaining the steady-state probabilities we only need to solve the system of linear equations:

$$\underline{\pi}\mathbf{Q} = \underline{0}, \quad \sum_{i\in\mathcal{I}} \pi_i = 1. \tag{21}$$

The right part (normalisation) is added to ensure that the obtained solution is indeed a probability vector; the left part alone has infinitely many solutions, which upon normalisation all yield the same probability vector.

Note that the equation $\underline{\pi}\mathbf{Q} = \underline{0}$ is of the same form as the equation $\underline{v} = \underline{v}\mathbf{P}$ we have seen for DTMCs. Since this latter equation can be rewritten as $\underline{v}(\mathbf{P}-\mathbf{I}) = \underline{0}$, the matrix $(\mathbf{P} - \mathbf{I})$, as already encountered in (11), can be interpreted as a generator matrix.

Note that we can also solve the steady-state probabilities of a CTMC by seeing it as a special case of an SMC, with embedded DTMC described by the probabilities $p_{i,j} = q_{i,j}/|q_{i,i}|$ and mean state residence times $h_i = |q_{i,i}|^{-1}$.

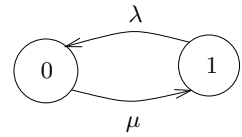


Fig. 5. A simple 2-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 4. Evaluation of a 2-state CTMC. Consider a component that is either operational or not. The time it is operational is exponentially distributed with mean $1/\lambda$. The time it is not operational is also exponentially distributed, with mean $1/\mu$. Signifying the operational state as state "1", and the down state as state "0", we can model this system as a 2-state CTMC with generator matrix $\mathbf{Q}$ as follows:

$$\mathbf{Q} = \begin{pmatrix} -\mu & \mu \\ \lambda & -\lambda \end{pmatrix}.$$

Furthermore, it is assumed that the system is initially fully operational so that $\underline{\pi}(0) = (0, 1)$. In Figure 5 we show the corresponding state-transition diagram. Solving (21) yields the following steady-state probability vector:

$$\underline{\pi} = \left(\frac{\lambda}{\lambda + \mu}, \frac{\mu}{\lambda + \mu} \right). \tag{22}$$

This probability vector can also be computed from the embedded DTMC which is given as:

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Solving for $\underline{v}$ yields us $\underline{v} = (\frac{1}{2}, \frac{1}{2})$, indicating that both states are visited equally often. However, these visits are not equally long. Incorporating the mean state residence times, being respectively $1/\mu$ and $1/\lambda$, yields

$$\underline{p} = \left(\frac{\frac{1}{2}\frac{1}{\mu}}{\frac{1}{2}\left(\frac{1}{\mu} + \frac{1}{\lambda}\right)}, \frac{\frac{1}{2}\frac{1}{\lambda}}{\frac{1}{2}\left(\frac{1}{\mu} + \frac{1}{\lambda}\right)} \right) = \left(\frac{\lambda}{\lambda + \mu}, \frac{\mu}{\lambda + \mu} \right), \tag{23}$$

which is the solution we have seen before.

For the transient behaviour of the CTMC we have to solve the corresponding system of linear differential equations. Although this is difficult in general, for this specific example we can obtain the solution explicitly. We obtain (see also Section 6):

$$\underline{\pi}(t) = \underline{\pi}(0) e^{\mathbf{Q}t}, \tag{24}$$

from which we can derive

$$\pi_0(t) = \frac{\lambda}{\lambda+\mu} - \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t},$$
$$\pi_1(t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t}. \tag{25}$$

Notice that $\pi_0(t) + \pi_1(t) = 1$ (for all t) and that the limit of the transient solutions for $t \to \infty$ indeed equals the steady-state probability vectors derived before. In Figure 6 we show the transient and steady-state behaviour of the 2-state CTMC for $3\lambda = \mu = 1$.

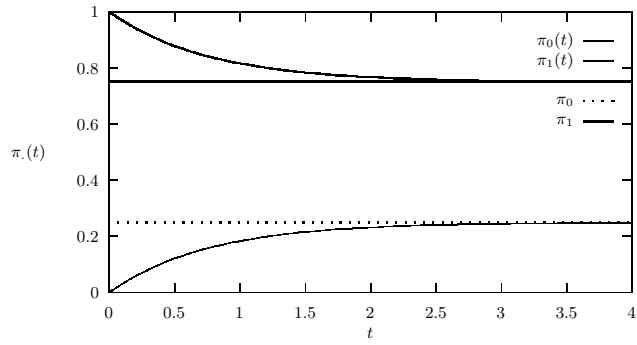


Fig. 6. Steady-state and transient behaviour of a 2-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 5. Availability evaluation of a fault-tolerant system. Consider a fault-tolerant computer system consisting of three computing nodes and a single voting node. The three computing nodes generate results after which the voter decides upon the correct value (by selecting the answer that is given by at least two computing nodes). Such a fault-tolerant computing system is also known as a triple-modular redundant system (TMR). The failure rate of a computing node is λ and of the voter ν failures per hour (fph). The expected repair time of a computing node is $1/\mu$ and of the voter is $1/\delta$ hours. If the voter fails, the whole system is supposed to have failed and after a repair (with rate δ) the system is assumed to start "as new". The system is assumed to be operational when at least two computing nodes and the voter are functioning correctly.

To model the availability of this system as a CTMC, we first have to define the state space: $\mathcal{I} = \{(3,1),(2,1),(1,1),(0,1),(0,0)\}$, where state (i,j) specifies that

i computing nodes are operational as well as j voters. Note that the circumstance of the computing nodes does not play a role any more as soon as the voter goes down; after a repair in this down state the whole system will be fully operational, irrespective of the past state. Using the above description, the state-transition diagram can be drawn easily, as given in Figure 7. The corresponding generator matrix is given as:

$$\mathbf{Q} = \begin{pmatrix} -(3\lambda+\nu) & 3\lambda & 0 & 0 & \nu \\ \mu & -(\mu+2\lambda+\nu) & 2\lambda & 0 & \nu \\ 0 & \mu & -(\mu+\lambda+\nu) & \lambda & \nu \\ 0 & 0 & \mu & -(\mu+\nu) & \nu \\ \delta & 0 & 0 & 0 & -\delta \end{pmatrix}. \tag{26}$$

We assume that the system is fully operational at $t = 0$. The following numerical parameters are given: $\lambda = 0.01$ fph, $\nu = 0.001$ fph, $\mu = 1.0$ repairs per hour (rph) and $\delta = 0.2$ rph.

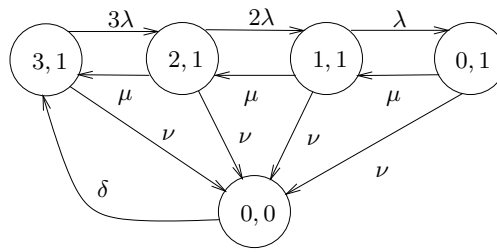


Fig. 7. CTMC for the TMR system (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

We can now compute the steady-state probabilities by solving the linear system $\underline{\pi}\mathbf{Q} = \underline{0}$ under the condition that $\sum_i \pi_i = 1$ (see Section 5) which yields the following values (note that we use the tuple (i, j) as state index here):

(i,j)	$(3,1)$	$(2,1)$	$(1,1)$	$(0,1)$	$(0,0)$
$\pi_{(i,j)}$	9.6551×10^{-1}	2.8936×10^{-2}	5.7813×10^{-4}	5.7755×10^{-6}	4.9751×10^{-3}

The probability that the system is operational can thus be computed as 0.99444. Although this number looks very good (it is very close to 100%) for a non-stop transaction processing facility, it would still mean an expected down-time of 48.7 hours a year ($(1 - 0.99444) \times 24 \times 365$).

The transient behaviour of this small CTMC can be obtained by numerically solving the differential equation for $\underline{\pi}(t)$ with a technique known as uniformisation (see Section 6). In Figure 8 we show the probability $\pi_{(3,1)}(t)$ for the first 10 hours of system operation. As can be observed, the transient probability reaches

the steady-state probability relatively fast. A similar observation can be made for the other transient probabilities in Figure 9 (note the logarithmic scale of the vertical axis).

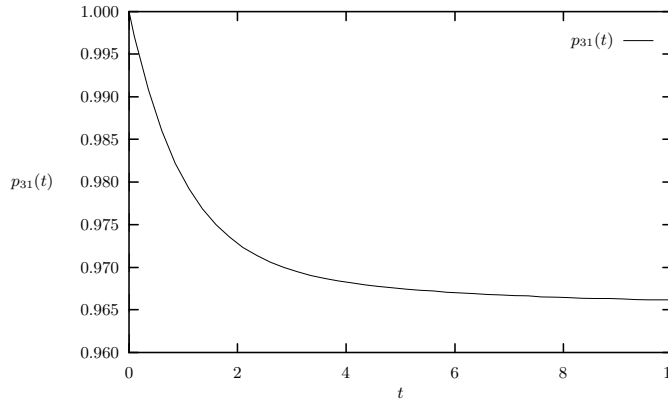


Fig. 8. Transient probability $\pi_{(3,1)}$ for the TMR system (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 6. A finite-buffer queueing station. Consider a single server that accepts requests to be processed in first-come first-served order. The processing time is assumed to be exponentially distributed with mean $1/\mu$ and the interarrival times are exponentially distributed with mean $1/\lambda$. An arrival process in which the interarrival times are independent and negative exponentially distributed is called a *Poisson process*. The number of arrivals taking place over a finite time interval $[0, t)$ in a Poisson process with rate λ follows a Poisson distribution with mean λt; $\Pr\{n \text{ arrivals in } [0,t)\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!}$, $n \in \mathbb{N}$, named after Professor Siméon Denis Poisson, who lived in France from 1781 through 1840. Being an excellent mathematician, he published largely over 300 articles, devoted to a wide variety of topics. His name is attached to a wide area of concepts, e.g., as in the probability-related examples above, but also in the Poisson integral, the Poisson equation for potential energy and Poisson's constant in electricity.

The state of the server is, due to the involved memoryless distributions, completely given by the number of requests in the server. If we assume that the server can hold at most K requests, (including the one actually being processed) the state of the server is governed by a CTMC, as given in Figure 10. In fact, we are dealing here with the CTMC underlying the so-called M|M|1|K queue, in which both the interarrival and service times are memoryless (explaining the two "M"s), hence negative exponentially distributed, there is 1 server and there

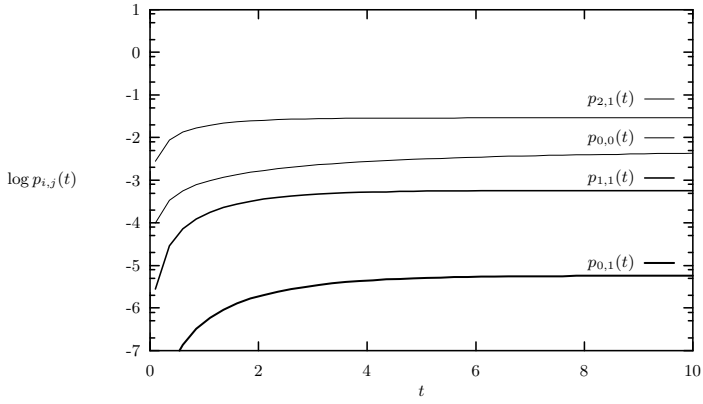


Fig. 9. Transient probabilities $\pi_{(i,j)}$ for the TMR system (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

are K buffer spots (including the server itself). The notation employed here to denote the particular queueing system is due to D.G. Kendall [43] (professor emeritus of Oxford University since 1989).

By the fact that the CTMC has a structure in which only left and right "neighbouring" states can be reached, this type of CTMC is called a *birth-death process*. The $(K+1)\times(K+1)$ generator matrix for the CTMC is given as follows:

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda & 0 & \cdots & \cdots \\ \mu & -(\lambda+\mu) & \lambda & 0 & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \mu & -\mu \end{pmatrix}.$$

The special *tridiagonal* structure of $\mathbf{Q}$ is typical for birth-death processes. Exploiting the birth-death structure of the CTMC, we can solve the equation $\underline{\pi}\mathbf{Q} = \underline{0}$ explicitly to reveal that $\pi_i = \pi_0 \cdot \rho,\ i = 1, \cdots, K$. Here, $\rho = \lambda/\mu$ is the ratio of the arrival rate and the service rate, which is also called the *traffic intensity* or the *utilisation*. We observe that all steady-state probabilities are related to the probability that the server is empty (π_0). The normalisation $\sum_i \pi_i = 1$ then yields π_0, in the following way:

$$\pi_0 \sum_{i=1}^{K} \rho^i = 1 \quad \Rightarrow \quad \pi_0 = \frac{1-\rho}{1-\rho^{K+1}},$$

where the latter equality follows from the *geometric series*: $\sum_{i=0}^{K} a^i = (1 - a^{K+1})/(1-a)$ (with $a > 0, a \neq 1$). Note at this point that only the steady-state probabilities can be obtained explicitly; the transient probabilities can only be obtained numerically.

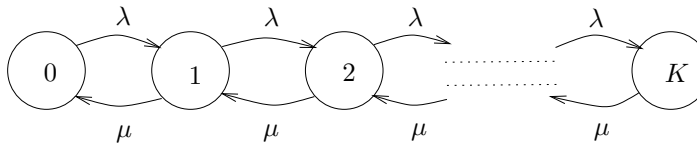


Fig. 10. CTMC underlying the M|M|1|K queue

Example 7. An infinite-buffer queueing station. We can extend the previous example by making the buffering capacity of the server unbounded. Surprisingly, a closed-form solution for the steady-state probabilities then still exists. The state space of the corresponding CTMC then equals $\mathbb{N}$ and we still have $\pi_i = \pi_0\rho$. Furthermore, if we require $\lambda < \mu$, i.e., the average number of requests arriving per unit of time is smaller than the average number of jobs that can be handled per unit of time, we have $\rho < 1$, so that π_i becomes smaller for increasing i. Moreover, the sum $\sum_{i=0}^{\infty} \rho^i = (1-\rho)^{-1}$, so that we find for all $i \in \mathbb{N}$: $\pi_i = (1-\rho)\rho^i$ (which is a geometric distribution). Furthermore, we can simply obtain a closed-form solution for the mean number of requests in the server:

$$E[N] = \sum_{i=0}^{\infty} i\pi_i = \sum_{i=0}^{\infty} i(1-\rho)\rho^i = \frac{\rho}{1-\rho}, \quad 0 \leq \rho < 1.$$

This example shows that, provided a regular structure exists in the Markov chain, steady-state probabilities can still be obtained explicitly, even if the state space is infinitely large. For more information on this topic, refer for instance to [27,67].

Example 8. Erlang's loss model. As stated in the introduction, Erlang studied Markovian models of telephone exchanges. In Kendall's notation, his model can now be described as an M|M|K|K queueing model, in which calls arrive according to a Poisson process with rate λ and take an exponentially distributed time with length $1/\mu$. Furthermore, the telephone switch considered can accommodate K simultaneous calls ("there are K lines") and cannot put calls on hold. Clearly, when all K lines are busy, an arriving call will be lost; the caller will hear a busy tone. The problem to be solved then, is to compute the required number of lines K so that, given traffic characteristics in terms of λ and μ, the call loss probability remains under some threshold.

Erlang's model describes a birth-death process, as illustrated in Figure 11, where the state number denotes the number of calls in progress. The call rate λ is constant for all states. The service rate linearly depends on the number of calls underway. For this birth-death process, the matrix $\mathbf{Q}$ has again a tridiagonal structure and we can easily solve the steady-state probabilities explicitly. Defining $\rho = \lambda/\mu$, we find:

$$\pi_i = \pi_0 \frac{\rho^i}{i!}, \quad i = 0, \cdots, K, \quad \text{with } \pi_0 = \left(\sum_{j=0}^{K} \frac{\rho^j}{j!} \right)^{-1},$$

where the expression for π_0 follows from the normalisation equation. The probability that an arriving call is lost, is now given by the probability for state K, that is:

$$\Pr\{\text{arriving call lost}; K, \rho\} = \frac{\rho^K / K!}{\sum_{i=0}^{K} \rho^i / i!}.$$

This result is also known as *Erlang's loss formula* $B(K, \rho)$. As part of his studies, Erlang published large tables with these loss probabilities, which were used to dimension telephone switches.

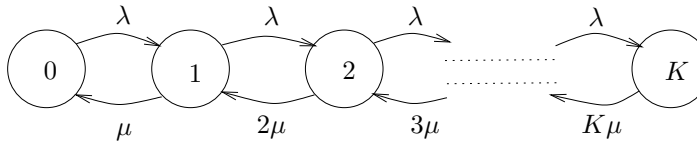


Fig. 11. CTMC underlying the M|M|K|K queue

5 Solution Methods for Steady-State Probabilities

As has become clear from the previous sections, in order to obtain the steady-state probabilities of finite DTMCs and CTMCs (with N states; numbered 1 through N) we need to solve a system of N linear equations which takes the following form (here given for a CTMC, but similar in the DTMC case):

$$\underline{\pi}\mathbf{Q} = \underline{0}, \quad \sum_{i=1}^{N} \pi_i = 1, \tag{27}$$

We assume here that the Markov chain is irreducible and aperiodic such that $\underline{\pi}$ does exist and is independent from $\underline{\pi}(0)$. Notice that the left part of (27) in fact does not uniquely define the steady-state probabilities; however, together with the normalisation equation a unique solution is found. For the explanations that follow, we will transpose the matrix $\mathbf{Q}$ and denote it as $\mathbf{A}$. Hence, we basically have to solve the following system of linear equations:

$$\mathbf{A}\underline{\pi}^T = \underline{b}, \quad \text{with} \quad \mathbf{A} = \mathbf{Q}^T \quad \text{and} \quad \underline{b} = \underline{0}^T. \tag{28}$$

Starting from this system of equations, two solution approaches can be chosen: *direct methods* or *iterative methods*. These methods will be discussed in Section 5.1 and 5.2, respectively.

5.1 Direct Methods

The main characteristic of a so-called direct method is that it aims at rewriting the system of equations in such a form that explicit expressions for the steady-state probabilities are obtained. The rewriting procedure costs an *a priori* known number of operations, given the number of states N.

Gaussian Elimination Perhaps the best-known direct solution technique is *Gaussian elimination*, named after the famous German mathematician Johann Carl Friedrich Gauss (1777–1855). The Gaussian elimination procedure consists of two phases: a reduction phase and a substitution phase.

In the *reduction phase* repetitive subtractions of equations from one another are used to make the system of equations upper-triangular (see also Figure 12). To do so, let the i-th equation be $\sum_j a_{i,j}p_j = 0$ (this equals $\sum_j p_j q_{j,i} = 0$ in the non-transposed system). We now vary i from 1 to N. The j-th equation, with $j = i+1, \cdots, N$, is now changed by subtracting the i-th equation $m_{j,i}$ times from it, where $m_{j,i} = a_{j,i}/a_{i,i}$, that is, we reassign the $a_{j,k}$ values as follows:

$$a_{j,k} := a_{j,k} - m_{j,i}a_{i,k}, \quad j,k > i.$$

Clearly, $a_{j,i} := a_{j,i} - m_{j,i}a_{i,i} = 0$, for all $j > i$. To avoid round-off errors, it is important to *set* $a_{j,i}$ to zero. By repeating this procedure for increasing i, the linear system of equations is transformed, in $N-1$ steps, to an upper-triangular system of equations. The element $a_{i,i}$ that acts as a divisor is called the *pivot*. If a pivot is encountered that equals 0, an attempt to divide by 0 results, which indicates that the system of equations does not have a solution. Since $\mathbf{Q}$ is a generator matrix of an irreducible ergodic CTMC, this problem will not occur. Moreover, since $\mathbf{A}$ is weakly diagonal dominant ($a_{i,i}$ is as large as the sum of all the values $a_{j,i}$ ($j \neq i$) in the same column) we have that $m_{j,i} < 1$ so that overflow problems are unlikely to occur.

At the end of the reduction phase, the N-th equation will always reduce to a trivial one ($0 = 0$). This is no surprise, since the system of equations without normalisation is not of full rank. We might even completely ignore the last equation. Since the right-hand side of the linear system of equations equals $\underline{0}$, nothing changes there either. When the right-hand side is a non-zero vector $\underline{b}$, we would have to set $b_j := b_j - m_{j,i}b_i$, for all $j > i$ in each step in the reduction process.

After the reduction has been performed, the *substitution phase* can start. The equation for π_N does not help us any further; we therefore assume a value $\alpha > 0$ for π_N, which can be substituted in the first $N-1$ equations, thus yielding a system of equations with one unknown less. We implement this by setting $b_j := b_j - a_{j,N}\pi_N$. Now, the $(N-1)$-th equation will have only one unknown left which we can directly compute as $\pi_{N-1} = b_{N-1}/a_{N-1,N-1}$. This new value can be substituted in the $N-2$ remaining equations, after which the $(N-2)$-th equation has only one unknown. This procedure can be repeated until all probabilities have been computed explicitly in terms of π_N (or α). We then use

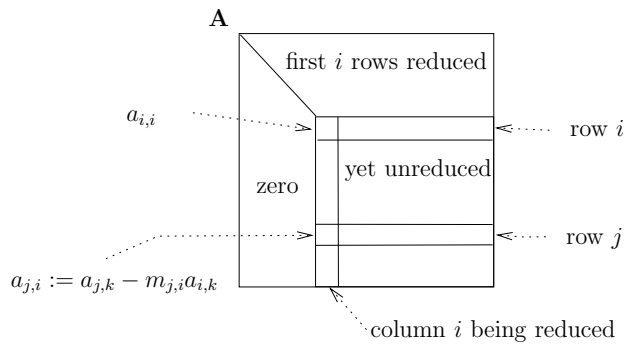


Fig. 12. Schematic representation of the i-th reduction step in the Gaussian elimination procedure (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

the normalisation equation to compute α to obtain the true probability vector, that is, we compute $\sigma = \sum_{i=1}^{N} \pi_i$ and set $\pi_i := \pi_i/\sigma$, for all i.

Instead of assuming the value α for π_N, we can also directly include the normalisation equation in the Gaussian elimination procedure. The best way to go then, is to replace the N-th equation with the equation $\sum_i \pi_i = 1$. In doing so, the last equation will directly give us π_N. The substitution phase can proceed as before.

Complexity Considerations for Gaussian Elimination The computational complexity for Gaussian elimination is $\mathcal{O}(N^3)$. By a more careful study of the algorithm, one will find that about $N^3/3 + N^2/2$ multiplications and additions have to be performed, as well as $N(N+1)/2$ divisions. Clearly, these numbers increase rapidly with increasing N. The main problem with Gaussian elimination, however, lies in its storage requirements. Although $\mathbf{A}$ will initially be sparse for most models, the reduction procedure normally increases the number of non-zeros in $\mathbf{A}$. At the end of the reduction phase, most entries of the upper half of $\mathbf{A}$ will be non-zero. The non-zero elements generated during this phase are called *fill-ins*. They can only be inserted efficiently when direct storage structures (arrays) are used. To store the upper-triangular matrix $\mathbf{A}$, $N^2/2$ floats have to be stored, each taking at least 8 bytes, plus 2 to 4 bytes for the corresponding indices. For moderately sized models, generated from some high-level model specification, N can easily be as large as 10^5 or even 10^6. This then precludes the use of Gaussian elimination. Fortunately, there are methods to compute $\underline{\pi}$ that do not change $\mathbf{A}$ and that are very fast as well. We will discuss these methods after we have discussed one alternative direct method.

LU Decomposition A method known as LU decomposition is advantageous to use when multiple systems of equations have to be solved, all of the form $\mathbf{A}\underline{x} = \underline{b}$, for different values of $\underline{b}$. This occurs, for instance when one tries to invert $\mathbf{A}$ by solving $\mathbf{A}\underline{x}_i = \underline{e}_i$, where the vectors $\underline{e}_i$ have as single nonzero element a 1 at the i-th position; the matrix $\mathbf{A}^{-1} = (\underline{x}_1, \underline{x}_2, \cdots)$.

The LU method starts by decomposing $\mathbf{A}$ such that it can be written as the *product* of two matrices $\mathbf{L}$ and $\mathbf{U}$, where the former is lower-triangular, and the latter is upper-triangular. We have:

$$\mathbf{A}\underline{x} = \underline{b} \quad \Rightarrow \quad \mathbf{L}\underbrace{\mathbf{U}\underline{x}}_{\underline{z}} = \underline{b}. \tag{29}$$

After the decomposition has taken place, we solve $\mathbf{L}\underline{z} = \underline{b}$, after which we solve $\mathbf{U}\underline{x} = \underline{z}$. Since the last two systems of equations are triangular, their solution can be found by a simple forward- and back-substitution.

The main question then lies in the computation of suitable matrices $\mathbf{L}$ and $\mathbf{U}$. Since $\mathbf{A}$ is the product of these two matrices, we know that

$$a_{i,j} = \sum_{k=1}^{N} l_{i,k} u_{k,j}, \quad i,j = 1, \cdots, N. \tag{30}$$

Given the fact that $\mathbf{L}$ and $\mathbf{U}$ are lower- and upper-triangular, we have to find $N^2 + N$ unknowns:

$$\begin{cases} l_{i,j}, & i = 1, \cdots, N, \ k = 1, \cdots, i, \\ u_{k,j}, & k = 1, \cdots, N, \ j = k, \cdots, N. \end{cases} \tag{31}$$

Since (30) only consists of N^2 equations, we have to assume N values to determine a unique solution. Two well-known schemes for this are [66]: (i) the Doolittle decomposition where one assumes $l_{i,i} = 1$, $i = 1, \cdots, N$; (ii) the Crout decomposition where one assumes $u_{i,i} = 1$, $i = 1, \cdots, N$.

Let us consider the Doolittle variant. First notice that in (30) many of the terms in the summation are zero, since one of the multiplicants is zero. In fact, we can rewrite (30) in a more convenient form as follows:

$$\begin{cases} i \leq j : a_{i,j} = u_{i,j} + \sum_{k=1}^{i-1} l_{i,k} u_{k,j}, \\ i > j : a_{i,j} = l_{i,j} u_{j,j} \sum_{k=1}^{j-1} l_{i,k} u_{k,j}. \end{cases} \tag{32}$$

From this system of equations, we can now iteratively compute the entries of $\mathbf{L}$ and $\mathbf{U}$ as follows:

$$\begin{cases} i \leq j : u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j}, \\ i > j : l_{i,j} = \frac{1}{u_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j} \right), \end{cases} \tag{33}$$

by increasing i from 1 until N is reached.

Example 9. LU decomposition after Doolittle. Suppose we want to decompose

$$\mathbf{A} = \begin{pmatrix} 3 & 2 & 5 \\ -6 & 1 & 8 \\ -7 & 2 & -3 \end{pmatrix},$$

using a Doolittle LU decomposition. We then know that

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ \cdot & 1 & 0 \\ \cdot & \cdot & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot \\ 0 & 0 & \cdot \end{pmatrix}.$$

We start to compute $u_{1,1} = a_{1,1} = 3$. We then compute $l_{2,1} = a_{2,1}/u_{1,1} = -2$ and $u_{1,2} = a_{1,2} = 2$. From this, we find $u_{2,2} = a_{2,2} - l_{2,1}u_{1,2} = 5$. We then compute $l_{3,1} = -\frac{7}{3}$ and find $l_{3,2} = \frac{4}{3}$. Via $u_{1,3} = a_{1,3} = 5$ and $u_{2,3} = 18$ we find $u_{3,3} = a_{3,3} - \sum_{k=1}^{2} l_{3,k}u_{k,3} = -\frac{46}{3}$. We thus have:

$$\mathbf{A} = \mathbf{LU} \quad \text{with} \quad \mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -2\frac{1}{3} & 1\frac{1}{3} & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} 3 & 2 & 5 \\ 0 & 5 & 18 \\ 0 & 0 & -15\frac{1}{3} \end{pmatrix}.$$

To solve $\mathbf{A}\underline{x} = \underline{1}$, we first solve for $\underline{z}$ in $\mathbf{L}\underline{z} = \underline{1}$. A simple substitution procedure yields $\underline{z} = (1, 3, -\frac{2}{3})$. We now continue to solve $\mathbf{U}\underline{x} = \underline{z}$; also here a substitution procedure suffices to find $\underline{x} = \frac{1}{115}(-4, 51, 5)$.

Example 10. LU decomposition for a CTMC. We reconsider the CTMC for which the matrix $\mathbf{Q}$ is given by

$$\mathbf{Q} = \begin{pmatrix} -4 & 2 & 2 \\ 1 & -2 & 1 \\ 6 & 0 & -6 \end{pmatrix}.$$

We form $\mathbf{A} = \mathbf{Q}^T$ and directly include the normalisation equation. To find the steady-state probabilities we thus have to solve:

$$\begin{pmatrix} -4 & 1 & 6 \\ 2 & -2 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{34}$$

We now decompose $\mathbf{A}$ using the Doolittle decomposition as follows:

$$\mathbf{A} = \mathbf{LU} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & -\frac{10}{12} & 1 \end{pmatrix} \begin{pmatrix} -4 & 1 & 6 \\ 0 & -\frac{3}{2} & 3 \\ 0 & 0 & 5 \end{pmatrix}. \tag{35}$$

The solution of $\mathbf{L}\underline{z} = (0, 0, 1)^T$ now reveals, via a simple substitution, that $\underline{z} = (0, 0, 1)$. We now have to find $\underline{\pi}$ from $\mathbf{U}\underline{\pi} = \underline{z}$, from which we, again via a substitution procedure, find $\underline{\pi} = (\frac{2}{5}, \frac{2}{5}, \frac{1}{5})$, as we have seen before.

In the above example, we took a specific way to deal with the normalisation equation: we replaced one equation from the "normal" system with the normalisation equation. In doing so, the vector $\underline{b}$ changes to $\underline{b} = (0, 0, 1)$ and after the solution of $\mathbf{L}\underline{z} = \underline{b}$, we found $\underline{z} = (0, 0, 1)^T$. This is not only true for the above example; if we replace the last equation, the vector $\underline{z}$ *always* has this value, so that we do not really have to solve the system $\mathbf{L}\underline{z} = (0, 0, 1)^T$. Hence, after the LU decomposition has been performed, we can directly solve $\underline{\pi}$ from $\mathbf{U}\underline{\pi} = (0, \cdots, 0, 1)^T$.

Opposed to the above variant, we can also postpone the normalisation. We then decompose $\mathbf{A} = \mathbf{Q}^T = \mathbf{LU}$, for which we will find that the last row of $\mathbf{U}$ contains only 0's. The solution of $\mathbf{L}\underline{z} = \underline{0}$ will then *always* yield $\underline{z} = \underline{0}$, so that we can immediately solve $\mathbf{U}\underline{\pi} = \underline{0}$. This triangular system of equations can easily be solved via a back-substitution procedure; however, we have to assume $\pi_N = \alpha$ and compute the rest of $\underline{\pi}$ relative to α as well. A final normalisation will then yield the ultimate steady-state probability vector $\underline{\pi}$.

Postponing the normalisation is preferred in most cases for at least two reasons: (i) it provides an implicit numerical accuracy test in that the last row of $\mathbf{U}$ should equal 0; and (ii) it requires less computations than the implicit normalisation since the number of non-zeros in the matrices that need to be handled is smaller. Of course, these advantages will become more apparent for larger values of N.

Complexity Considerations for LU Decomposition The LU decomposition solution method has the same computational complexity of $\mathcal{O}(N^3)$ as the Gaussian elimination procedure. The decomposition can be performed with only one data structure (typically an array). Initially, the matrix $\mathbf{A}$ is stored in it, but during the decomposition the elements of $\mathbf{L}$ (except for the diagonal elements from $\mathbf{L}$, but these are equal to 1 anyway) and the elements of $\mathbf{U}$ replace the original values.

Under- and Overflow We finally comment on the occurrence of over- and underflow during the computations. Underflow can be dealt with by setting intermediate values smaller than some threshold, say 10^{-24}, equal to 0. Overflow is unlikely to occur during the reduction phase in the Gaussian elimination since the pivots are the largest (absolute) quantities in every column. If in other parts of the algorithms overflow tends to occur, which can be observed if some of the values grow above a certain threshold, e.g., 10^{10}, then an intermediate normalisation of the solution vector is required. A final normalisation then completes the procedures.

5.2 Iterative Methods

Although direct methods are suitable to solve the system of equations (28), for reasons of computational and memory efficiency they cannot be used when the number of states N grows beyond about a thousand. Instead, we use iterative

methods in these cases. With iterative methods, the involved matrices do not change (fill-in is avoided), so that they can be stored efficiently using sparse matrix methods. Moreover, these methods can be implemented such that in the matrix-multiplications only the multiplications involving *two* non-zero operands are taken into account.

Iterative procedures do not result in an explicit solution of the system of equations. A key characteristic of iterative methods is that it is not possible to state *a priori* how many computational steps are required. Instead, a simple numerical procedure (the iteration step) is performed repeatedly until a desired level of accuracy is reached.

The Power Method We have already seen the simplest iterative method to solve for the steady-state probabilities of a DTMC in Section 2: the *Power method*. The Power method performs successive multiplication of the steady-state probability vector $\underline{v}$ with $\mathbf{P}$ until convergence is reached. The Power method can also be applied to CTMCs. Given a CTMC with generator matrix $\mathbf{Q}$, we can compute the DTMC transition matrix $\mathbf{P} = \mathbf{I} + \mathbf{Q}/\lambda$. If we take $\lambda \geq \max_i\{|q_{i,i}|\}$, the matrix $\mathbf{P}$ is a stochastic matrix and describes the evolution of the CTMC in time-steps of mean length $1/\lambda$ (see Section 6 for a more precise formulation). Using $\mathbf{P}$ and setting $\underline{\pi}^{(0)} = \underline{\pi}(0)$ as initial estimate for the steady-state probability vector, we can compute $\underline{\pi}^{(k+1)} = \underline{\pi}^{(k)}\mathbf{P}$ and find that $\underline{\pi} = \lim_{k\to\infty} \underline{\pi}^{(k)}$.

In practice, the Power method is not very efficient. Since more efficient methods do exist, we do not discuss the Power method any further.

The Jacobi Method Two of the best-known (and simple) iterative methods are the Jacobi and the Gauss-Seidel iterative methods. For these methods, one first rewrites the i-th equation of the linear system (28) in the following way:

$$\sum_{j=1}^{N} a_{i,j}\pi_j = 0 \;\Rightarrow \pi_i = -\frac{1}{a_{i,i}}\left(\sum_{j<i} \pi_j a_{i,j} + \sum_{j>i} \pi_j a_{i,j}\right).$$

We clearly need $a_{i,i} \neq 0$; when the linear system is used to solve for the steady-state probabilities of an irreducible aperiodic Markov chain, this is guaranteed.

The iterative procedures now proceed with assuming a first guess for $\underline{\pi}$, denoted $\underline{\pi}^{(0)}$. If one does know an approximate solution for $\underline{\pi}$, it can be used as initial guess. In other cases, the uniform distribution is a reasonable choice, i.e., $\pi_i^{(0)} = 1/N$. The next estimate for $\underline{\pi}$ is then computed as follows:

$$\pi_i^{(k+1)} = -\frac{1}{a_{i,i}}\left(\sum_{j\neq i} \pi_j^{(k)} a_{i,j}\right). \tag{36}$$

This is the *Jacobi iteration* scheme. We continue to iterate until two successive estimates for $\underline{\pi}$ differ less than some ϵ from one another, i.e., when $||\underline{\pi}^{(k+1)} - \underline{\pi}^{(k)}|| < \epsilon$ (difference criterion). Notice that when this difference is very small,

this does *not always* imply that the solution vector has been found. Indeed, it might be the case that the convergence towards the solution is very slow. Therefore, it is good to check whether $||\mathbf{A}\underline{\pi}^{(k)}|| < \epsilon$ (residual criterion). Since this way of checking convergence is more expensive, often a combination of these two methods is used: use the difference criterion normally; once it is satisfied use the residual criterion. If the convergence is really slow, two successive iterates might be very close to one another, although the actual value for $\underline{\pi}$ is still "far away". To avoid the difference criterion to stop the iteration process too soon, one might instead check on the difference between non-successive iterates, i.e., $||\underline{\pi}^{(k)} - \underline{\pi}^{(k-d)}|| < \epsilon$, with $d \in \mathbb{N}^+$ (and $d \leq k$).

The Gauss-Seidel Method The Jacobi method requires the storage of both $\underline{\pi}^{(k)}$ and $\underline{\pi}^{(k+1)}$ during an iteration step. If, instead, the computation is structured such that the $(k+1)$-th estimates are used as soon as they have been computed, we obtain the *Gauss-Seidel* scheme:

$$\pi_i^{(k+1)} = -\frac{1}{a_{i,i}} \left(\sum_{j<i} \pi_j^{(k+1)} a_{i,j} + \sum_{j>i} \pi_j^{(k)} a_{i,j} \right), \tag{37}$$

where we assume that the order of computation is from $\pi_1^{(k+1)}$ to $\pi_N^{(k+1)}$. This scheme then requires only one probability vector to be stored, since the $(k+1)$-th estimate for π_i immediately replaces the k-th estimate in the single stored vector.

The SOR Method The last method we mention is the *successive over-relaxation method* (SOR). SOR is an extension of the Gauss-Seidel method, in which the vector $\underline{\pi}^{(k+1)}$ is computed as the weighted average of the vector $\underline{\pi}^{(k)}$ and the vector $\underline{\pi}^{(k+1)}$ that would have been used in the (pure) Gauss-Seidel iteration. That is, we have, for $i = 1, \cdots, N$:

$$\pi_i^{(k+1)} = (1-\omega)\pi_i^{(k)} - \frac{\omega}{a_{i,i}} \left(\sum_{j<i} \pi_j^{(k+1)} a_{i,j} + \sum_{j>i} \pi_j^{(k)} a_{i,j} \right),$$

where $\omega \in (0,2)$ is the relaxation factor. When $\omega = 1$, this method reduces to the Gauss-Seidel iteration scheme; however, when we take $\omega > 1$ (or $\omega < 1$) we speak over over-relaxation (under-relaxation). With a proper choice of ω, the iterative solution process can be accelerated significantly. Unfortunately, the optimal choice of ω cannot be determined *a priori*. We can, however, estimate ω during the solution process itself; for details, refer to Stewart [66] or Hageman and Young [26].

Example 11. Comparing the Power, the Jacobi and the Gauss-Seidel method. We reconsider the CTMC for which the matrix $\mathbf{Q}$ is given by

$$\mathbf{Q} = \begin{pmatrix} -4 & 2 & 2 \\ 1 & -2 & 1 \\ 6 & 0 & -6 \end{pmatrix}.$$

As starting vector for the iterations we take $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. In the Jacobi and Gauss-Seidel method we renormalised the probability vector after every iteration. In Table 1 we show the first ten iteration vectors for these methods. As can be seen, the Power method convergest slowest, followed by the Jacobi and the Gauss-Seidel method.

#	Power	Jacobi	Gauss-Seidel
1	(0.5000, 0.3333, 0.1667)	(0.5385, 0.3077, 0.1538)	(0.5833, 0.5833, 0.2917)
2	(0.3889, 0.3889, 0.2222)	(0.4902, 0.3137, 0.1961)	(0.4000, 0.4000, 0.2000)
3	(0.4167, 0.3889, 0.1944)	(0.3796, 0.4213, 0.1991)	(0.4000, 0.4000, 0.2000)
4	(0.3981, 0.3981, 0.2037)	(0.3979, 0.4023, 0.1998)	:
5	(0.4028, 0.3981, 0.1991)	(0.4001, 0.3999, 0.2000)	:
6	(0.3997, 0.3997, 0.2006)	(0.4000, 0.4000, 0.2000)	:
7	(0.4005, 0.3997, 0.1998)	(0.4000, 0.4000, 0.2000)	:
8	(0.3999, 0.3999, 0.2001)	(0.4000, 0.4000, 0.2000)	:
9	(0.4001, 0.3999, 0.2000)	:	:
10	(0.4000, 0.4000, 0.2000)	:	:

Table 1. The first few iteration vectors for three iterative solution methods (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Complexity Considerations Iterative methods can be used to solve the linear systems arising in the solution of the steady-state probabilities for Markov chains, with or without the normalisation equation. Quite generally we can state that it is better *not* to include the normalisation equation; if the normalisation equation is included, the second largest eigenvalue of the coefficient matrix $\mathbf{A}$ generally increases (the largest one is 1) which normally reduces the speed of convergence.

All iterative methods require the storage of the matrix $\mathbf{A}$. For larger modelling problems, $\mathbf{A}$ has to be stored sparsely; it is then important that the sparse storage structure is structured such that row-wise access is very efficient since all methods require the product of a row of $\mathbf{A}$ with the (column) iteration vector $\underline{\pi}^{(k)}$. The Power and the Jacobi method require two iteration vectors to be stored, each of length N. The Gauss-Seidel and the SOR method only require one such vector. In all the iteration schemes the divisions by $-a_{i,i}$ (and for SOR the multiplication with ω) need to be done only once, either before the actual iteration process starts or during the first iteration step, by changing the matrix $\mathbf{A}$ accordingly. This saves N divisions (and N multiplications for SOR) per iteration. A single iteration can then be interpreted as a single matrix-vector multiplication (MVM). In a non-sparse implementation, a single MVM costs $\mathcal{O}(N^2)$ multiplications and additions. However, in a suitably chosen sparse storage structure only $\mathcal{O}(\eta)$ multiplications and additions are required, where η is

the number of non-zero elements in **A**. Typically, the number of nonzero elements per column in **A** is limited to a few dozen. For example, if the CTMC is derived from a high-level model specification, the number of nonzero elements per row in **Q** equals the number of enabled activities in a particular state. This number is normally much smaller than N. Hence, it is reasonable to assume that one iteration step only takes $\mathcal{O}(N)$ operations.

An important difference between iterative methods is the number of required iterations. Typically, the Power method converges slowest, and the Gauss-Seidel method typically outperforms the Jacobi method. With the SOR method, a proper choice of the relaxation factor ω accelerates the iteration process, so that it often is the fastest method. In practical modelling problems, the required number of iterations can range from just a few to a few thousands.

There do exist more advanced methods to solve linear systems of equations which often convergence in less iteration steps. This then mostly comes at the cost of either more complex iteration steps (more computation time required per step) or iteration steps requiring much more intermediate solution vectors, or both. A fast method, for instance, requiring 7 iteration vectors is CGS (conjugate gradient squared), an example of a so-called Krylov subspace method [66, Chapter 4]. It goes beyond the scope of the current paper to go in more detail here.

6 Solution Methods for Transient-State Probabilities

In this section, we discuss the solution of the time-dependent behaviour of Markov chains. As we have seen in Section 2, the time-dependent behaviour of a DTMC is simply obtained by successive matrix-vector multiplications and is therefore not further considered here. The time-dependent behaviour of an SMC is much more complex; it goes beyond the scope of this paper. Hence, we focus on the transient behaviour of CTMCs in this section.

In Section 6.1 we explain why transient behaviour is of interest and which equations we need to solve for that purpose. We discuss "traditional" methods to solve these equations in Section 6.2 and continue with the use of uniformisation in Section 6.3. Finally, in Section 6.4, we comment on the use of uniformisation to compute so-called cumulative measures.

6.1 Introduction

Steady-state measures (probabilities) do suffice for the evaluation of the performance of most systems. There are, however, exceptions to this rule, for instance

- when the system life-time is so short that steady-state is not reached;
- when the period towards the steady-state situation itself is of interest;
- when temporary overload periods, for which no steady-state solution exists, are of interest;

– when reliability and availability properties are taken into account in the model, e.g., non-repairable systems that are failure-prone are of no interest in steady-state, since then they will have completely failed.

The time-dependent state probabilities of a CTMC are specified by a linear system of differential equations (as already given in (20)):

$$\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}, \quad \text{given} \quad \underline{\pi}(0). \tag{38}$$

Measures that are specified in terms of $\underline{\pi}(t)$ are called *instant-of-time measures*. If we associate a *reward* r_i with every state, the expected reward at time t can be computed as

$$E[X(t)] = \sum_{i=1}^{N} r_i \pi_i(t). \tag{39}$$

The rewards express the amount of gain (or costs) that is accumulated per unit of time in state i; $E[X(t)]$ then expresses the speed of gain accumulation (per time-unit).

In many modelling applications, not only the values of the state probabilities at a time instance t are of importance, but also the total time spent in any state up to some time t, as expressed in so-called *cumulative measures*. We define the cumulative state vector $\underline{l}(t)$ as

$$\underline{l}(t) = \int_0^t \underline{\pi}(s)ds. \tag{40}$$

Notice that the entries of $\underline{l}(t)$ are no longer probabilities; $l_i(t)$ denotes the overall time spent in state i during the interval $[0, t)$. Integrating (38), we obtain

$$\int_0^t \underline{\pi}'(s)ds = \int_0^t \underline{\pi}(s)\mathbf{Q}ds, \tag{41}$$

which can be rewritten as

$$\underline{\pi}(t) - \underline{\pi}(0) = \underline{l}(t)\mathbf{Q}, \tag{42}$$

which can, after having substituted $\underline{l}'(t) = \underline{\pi}(t)$, be written as

$$\underline{l}'(t) = \underline{l}(t)\mathbf{Q} + \underline{\pi}(0). \tag{43}$$

hence, $\underline{l}(t)$ follows from the solution of a linear non-homogeneous system of differential equations. If r_i is the reward obtained per time-unit in state i, then

$$Y(t) = \sum_{i=1}^{N} r_i l_i(t) \tag{44}$$

expresses the total amount of reward gained over the period $[0, t)$. The distribution $F_Y(y, t) = \Pr\{Y(t) \leq y\}$ has been defined by Meyer as the *performability distribution* [50,51]; it expresses the probability that a reward of at most y is gained in the period $[0, t)$. Meyer developed his performability measure in order to express the effectiveness of use of computer systems in failure prone environments.

Example 12. Measure interpretation. Consider a three-state CTMC with generator matrix

$$\mathbf{Q} = \begin{pmatrix} -2f & 2f & 0 \\ r & -(f+r) & f \\ 0 & r & -r \end{pmatrix}.$$

This CTMC models the availability of a computer system with two processors. In state 1 both processors are operational but can fail with rate $2f$. In state 2 only one processor is operational (and can fail with rate f); the other one is repaired with rate r. In state 3 both processors have failed; one of them is being repaired. Note that we assume that both the processor life-times and the repair times are negative exponentially distributed. Since in state 1 both processors operate, we assign a reward 2μ to state 1, where μ is the effective processing rate of a single processor. Similarly, we assign $r_2 = \mu$ and $r_3 = 0$. We assume that the system is initially fully operational, i.e., $\underline{\pi}(0) = (1, 0, 0)$. The following measures can now be computed:

- Steady-state reward rate $\sum_i r_i \pi_i$: the expected processing rate of the system in steady-state, i.e., the long-term average processing rate of the system;
- Expected instant reward rate $\sum_i r_i \pi_i(t)$: the expected processing rate at a particular time instance t;
- Expected accumulated reward $\sum_i r_i l_i(t)$: the expected number of jobs (of length 1) processed in the interval $[0, t)$;
- Accumulated reward distribution $F_Y(y, t)$: the probability that at most y jobs (of length 1) have been processed during $[0, t)$.

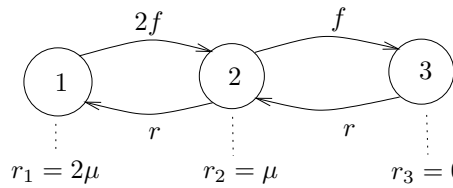


Fig. 13. A three-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

6.2 Runge-Kutta Methods

The numerical solution of systems of differential equations of type (38) and (43) has since long been an important topic in numerical mathematics. Many numerical procedures have been developed for this purpose, all with specific

strengths and weaknesses. Below, we will present one such method in a concise way, thereby focusing on the computation of $\underline{\pi}(t)$; for details, see [66].

With *Runge-Kutta methods* (RK-methods) the continuous vector function $\underline{\pi}(t)$ that follows from the differential equation $\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}$, given $\underline{\pi}(0)$, is approximated by a discrete function $\underline{\tilde{\pi}}_i$ $(i \in \mathbb{N})$, where $\underline{\tilde{\pi}}_i \approx \underline{\pi}(ih)$, i.e., h is the fixed step-size in the discretisation; the smaller h, the better (but more expensive) the solution.

With RK-methods, the last computed value for any point $\underline{\tilde{\pi}}_i$ is used to compute $\underline{\tilde{\pi}}_{i+1}$. The values $\underline{\tilde{\pi}}_0$ through $\underline{\tilde{\pi}}_{i-1}$ are not used to compute $\underline{\tilde{\pi}}_{i+1}$. For this reason, RK-methods are called *single-step methods*. They are always stable, provided the step-size h is taken sufficiently small. Unlike Euler-methods, RK-methods do not require the computation of derivatives of the function of interest, which keeps them fairly efficient. RK-methods are distinguished on the basis of their *order*: a RK-method is of order k if the exact Taylor series for $\underline{\pi}(t+h)$ and the solution of the RK-scheme for time instance $t+h$ coincide as far as the terms up to h^k are concerned.

One of the most widely used RK-methods is the 4th-order RK-method (normally denoted as "RK4"). For a vector-differential equation $\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}$, given $\underline{\pi}(0)$, successive estimates for $\underline{\tilde{\pi}}_i$ are computed as follows:

$$\underline{\tilde{\pi}}_{i+1} = \underline{\tilde{\pi}}_i + \frac{h}{6}(\underline{k}_1 + 2\underline{k}_2 + 2\underline{k}_3 + \underline{k}_4), \tag{45}$$

with

$$\begin{cases} \underline{k}_1 = \underline{\tilde{\pi}}_i \mathbf{Q}, \\ \underline{k}_2 = (\underline{\tilde{\pi}}_i + \frac{h}{2}\underline{k}_1)\mathbf{Q}, \\ \underline{k}_3 = (\underline{\tilde{\pi}}_i + \frac{h}{2}\underline{k}_2)\mathbf{Q}, \\ \underline{k}_4 = (\underline{\tilde{\pi}}_i + h\underline{k}_3)\mathbf{Q}. \end{cases} \tag{46}$$

Since the RK4 method provides an explicit solution to $\underline{\tilde{\pi}}_i$, it is called an *explicit 4th-order method*. Per iteration step of length h, it requires 4 matrix-vector multiplications, 7 vector-vector additions and 4 scalar-vector multiplications. Furthermore, apart from $\mathbf{Q}$ and $\underline{\tilde{\pi}}$ also storage for at least two intermediate probability vectors is required.

In contrast, *implicit* RK-methods yield a system of linear equations in which the vector $\underline{\tilde{\pi}}_i$ appears implicitly. Such methods are normally more expensive to employ and can therefore only be justified in special situations, e.g., when the CTMC under study is stiff, meaning that the ratio of the largest and smallest rate appearing in $\mathbf{Q}$ is very large, say of the order of 10^4 or higher.

6.3 Uniformisation for Transient Measures

Consider the *scalar* differential equation $p'(t) = p(t)Q$, given $p(0)$ and scalar constant Q. From elementary analysis we know that the solution to this differential equation is $p(t) = p(0)e^{Qt}$. When dealing with CTMCs, the transient behaviour

is defined by the linear system of differential equations (20); the transient behaviour then can still be computed as an exponential, however, now in terms of vectors and matrices, that is,

$$\underline{\pi}(t) = \underline{\pi}(0)e^{\mathbf{Q}t}. \tag{47}$$

Direct computation of this matrix exponential, e.g., via a Taylor/MacLaurin series expansion as $\sum_{i=0}^{\infty}(\mathbf{Q}t)^i/i!$, is in general not a good idea [52]: (i) the infinite summation that appears in the Taylor series cannot be truncated efficiently; (ii) severe round-off errors usually will occur due to the fact that $\mathbf{Q}$ contains positive as well as negative entries; and (iii) the matrices $(\mathbf{Q}t)^i$ become non-sparse, thus requiring too much storage capacity for practically relevant applications. To avoid these problems, a method known as *uniformisation*, also known as Jensen's method or randomisation, is regarded as the method of choice [24,25,39]. To use uniformisation, we define the matrix

$$\mathbf{P} = \mathbf{I} + \frac{\mathbf{Q}}{\lambda} \quad \Rightarrow \quad \mathbf{Q} = \lambda(\mathbf{P} - \mathbf{I}). \tag{48}$$

If λ is chosen such that $\lambda \geq \max_i\{|q_{i,i}|\}$, then the entries in $\mathbf{P}$ are all between 0 and 1, whereas the rows of $\mathbf{P}$ sum to 1. In other words, $\mathbf{P}$ is a stochastic matrix and describes a DTMC. The value of λ is called *uniformisation rate*.

Example 13. Uniformising a CTMC. Consider the CTMC given by

$$\mathbf{Q} = \begin{pmatrix} -4 & 2 & 2 \\ 1 & -2 & 1 \\ 6 & 0 & -6 \end{pmatrix}. \tag{49}$$

and initial probability vector $\underline{\pi}(0) = (1, 0, 0)$. For the uniformisation rate we find by inspection: $\lambda = 6$, so that the corresponding DTMC is given by:

$$\mathbf{P} = \frac{1}{6}\begin{pmatrix} 2 & 2 & 2 \\ 1 & 4 & 1 \\ 6 & 0 & 0 \end{pmatrix}. \tag{50}$$

The CTMC and the DTMC are given in Figure 14.

The process of uniformising a CTMC can be understood as follows. In the CTMC, the state residence times are exponentially distributed. The state with the shortest residence times provides us with the uniformisation rate λ. For that state, one epoch in the DTMC corresponds to one negative exponentially distributed delay with rate λ, after which one of the successor states is selected probabilistically. For the states in the CTMC that have total outgoing rate λ, the corresponding states in the DTMC will have no self-loops. For states in the CTMC having a state residence time distribution with a rate smaller than λ (these states have on average a longer state residence time), one epoch in the DTMC might not be long enough; hence, in the next epoch these states might

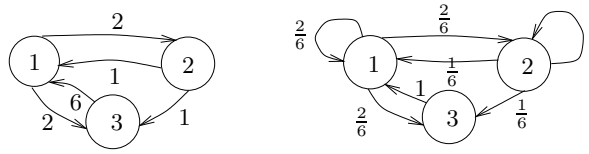


Fig. 14. A small CTMC (left) and the corresponding DTMC (right) after uniformisation (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

be revisited. This is made possible by the definition of $\mathbf{P}$, in which these states have self-loops, i.e., $p_{i,i} > 0$. Using (48) we can write

$$\underline{\pi}(t) = \underline{\pi}(0)e^{\mathbf{Q}t} = \underline{\pi}(0)e^{\lambda(\mathbf{P}-\mathbf{I})t} = \underline{\pi}(0)e^{-\lambda\mathbf{I}t}e^{\lambda\mathbf{P}t} = \underline{\pi}(0)e^{-\lambda t}e^{\lambda\mathbf{P}t}. \tag{51}$$

We now employ a Taylor-series expansion for the matrix exponential as follows:

$$\underline{\pi}(t) = \underline{\pi}(0)e^{-\lambda t}\sum_{n=0}^{\infty}\frac{(\lambda t)^n\mathbf{P}^n}{n!} = \underline{\pi}(0)\sum_{n=0}^{\infty}\psi(\lambda t;n)\mathbf{P}^n, \tag{52}$$

where

$$\psi(\lambda t;n) = e^{-\lambda t}\frac{(\lambda t)^n}{n!}, \quad n \in I\!N, \tag{53}$$

are Poisson probabilities, i.e., $\psi(\lambda t;n)$ is the probability of n events occurring in $[0,t)$ in a Poisson process with rate λ. Of course, we still deal with a Taylor series approach here, however, the involved $\mathbf{P}$-matrix is a probabilistic matrix with all its entries between 0 and 1, as are the Poisson probabilities. Hence, this Taylor series "behaves nicely", as we will discuss below.

Equation (52) can be understood as follows. At time t, the probability mass of the CTMC, initially distributed according to $\underline{\pi}(0)$ has been redistributed according to the DTMC with state-transition matrix $\mathbf{P}$. During the time interval $[0,t)$, with probability $\psi(\lambda t;n)$ exactly n jumps have taken place. The effect of these n jumps on the initial distribution $\underline{\pi}(0)$ is described by the vector-matrix product $\underline{\pi}(0)\mathbf{P}^n$. Weighting this vector with the associated Poisson probability $\psi(\lambda t;n)$, and summing over all possible numbers of jumps in $[0,t)$, we obtain, by the law of total probability, the probability vector $\underline{\pi}(t)$.

Uniformisation allows for an iterative solution without matrix-matrix multiplications, so that matrix fill-in do not occur. Instead of directly computing the n-th Power of $\mathbf{P}$ as suggested by (52) one considers the following sum of vectors:

$$\underline{\pi}(t) = \sum_{n=0}^{\infty}\psi(\lambda t;n)\,(\underline{\pi}(0)\mathbf{P}^n) = \sum_{n=0}^{\infty}\psi(\lambda t;n)\hat{\underline{\pi}}(n), \tag{54}$$

where $\hat{\underline{\pi}}_n$ is the state probability distribution vector after n epochs in the DTMC with transition matrix $\mathbf{P}$, which can be derived recursively as

$$\hat{\underline{\pi}}(0) = \underline{\pi}(0) \quad \text{and} \quad \hat{\underline{\pi}}(n) = \hat{\underline{\pi}}(n-1)\mathbf{P}, \quad n \in I\!N^+. \tag{55}$$

Clearly, the infinite sum in (54) has to be truncated, say after k_ϵ epochs in the DTMC. The actually computed state probability vector $\underline{\tilde{\pi}}(t)$ then equals:

$$\underline{\tilde{\pi}}(t) = \sum_{n=0}^{k_\epsilon} \psi(\lambda t; n)\underline{\hat{\pi}}(n). \tag{56}$$

The number of terms that has to be added to reach a prespecified accuracy ϵ can now be computed *a priori* as follows. It can be shown that the difference between the computed and the exact value of the transient probability vector is bounded as follows:

$$||\underline{\pi}(t) - \underline{\tilde{\pi}}(t)|| \leq 1 - \sum_{n=0}^{k_\epsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!}. \tag{57}$$

Thus, we have to find that value of k_ϵ such that $1 - \sum_{n=0}^{k_\epsilon} e^{-\lambda t}(\lambda t)^n/n! \leq \epsilon$. Stated differently, we need the smallest value of k_ϵ that satisfies

$$\sum_{n=0}^{k_\epsilon} \frac{(\lambda t)^n}{n!} \geq \frac{1-\epsilon}{e^{-\lambda t}} = (1-\epsilon)e^{\lambda t}. \tag{58}$$

For reasons that will become clear below, k_ϵ is called the *right truncation point*.

Example 14. How large should we take k_ϵ? In Table 2 we show the number of required steps k_ϵ as a function of ϵ and the product λt in the uniformisation procedure. As can be observed, k_ϵ increases sharply with increasing λt and decreasing ϵ.

If the product λt is large, k_ϵ tends to be of order $\mathcal{O}(\lambda t)$. On the other hand, if λt is large, the DTMC described by $\mathbf{P}$ might have reached steady-state along the way, so that the last matrix-vector multiplications do not need to be performed any more. Such a steady-state detection can be integrated in the computational procedure (see [57] and the example below).

	λt						
ϵ	0.1	0.2	1	2	4	8	16
0.0005	2	3	6	8	12	19	31
0.00005	3	3	7	10	14	21	34
0.000005	3	4	8	11	16	23	37

Table 2. The number of required steps k_ϵ as a function of ϵ and the product λt (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 15. Transient solution of a three-state CTMC. We consider the transient solution of the CTMC given in Figure 14; we already performed the uniformisation to form the matrix $\mathbf{P}$ with uniformisation rate $\lambda = 6$.

We first establish how many steps we have to take into account for increasing t. This number can be computed by checking the inequality (58) and taking $\epsilon = 10^{-4}$. We find:

t	0.1	0.2	0.5	1	5	10	20	50	100
k_ϵ	5	7	11	17	52	91	163	367	693

We then continue to compute $\underline{\tilde{\pi}}(t)$ according to (56) to find the curves for $\pi_i(t)$ as indicated in Figure 15. As can be observed, for $t \geq 2$ steady-state is reached. Although for larger values of t we require very many steps to be taken, the successive vectors $\underline{\hat{\pi}}(n)$ do not change any more. Denote with $k_{\text{ss}} < k_\epsilon$ the value after which $\underline{\hat{\pi}}_i$ does not change any more. Instead of explicitly computing the sum (56) for all values of n, the last part of it can then be computed more efficiently as follows:

$$\underline{\tilde{\pi}}(t) = \sum_{n=0}^{k_\epsilon} \psi(\lambda t; n)\underline{\hat{\pi}}(n) = \sum_{n=0}^{k_{\text{ss}}} \psi(\lambda t; n)\underline{\hat{\pi}}(n) + \underbrace{\left(\sum_{n=k_{\text{ss}}+1}^{k_\epsilon} \psi(\lambda t; n)\right)}_{1-\sum_{n=0}^{k_{\text{ss}}} \psi(\lambda t;n)} \underline{\hat{\pi}}(k_{\text{ss}}), \quad (59)$$

thus saving the computation intensive matrix-vector multiplications in the last part of the sum. The point k_{ss} is called the *steady-state truncation point*.

If the product λt is very large, the first group of Poisson probabilities is very small, often so small that the corresponding vectors $\underline{\hat{\pi}}(n)$ do not really matter. We can exploit this by only starting to add the weighted vectors $\underline{\hat{\pi}}(n)$ after the Poisson weighting factors become reasonably large. Of course, we still have to compute the matrix-vector products (55). The point where we start to add the probability vectors is called the *left truncation point*.

Finally, we note that the Poisson probabilities $\psi(\lambda t; n), n = 0, \cdots, N$, can be computed efficiently when taking into account the following recursive relations:

$$\psi(\lambda t; 0) = e^{-\lambda t}, \quad \text{and} \quad \psi(\lambda t; n+1) = \psi(\lambda t; n)\frac{\lambda t}{n+1}, \quad n \in I\!N. \quad (60)$$

When λt is large, say larger than 25, overflow might easily occur. However, for these cases, the normal distribution can be used as an approximation. Fox and Glynn report on a stable algorithm to compute Poisson probabilities [21].

To use uniformisation, the sparse matrix $\mathbf{P}$ has to be stored, as well as two probability vectors. Given an N-state Markov chain, two probability vectors of length N have to be stored. Given that the matrix $\mathbf{P}$ is sparse, which typically is the case, the cost to store it is of order N. Hence, the overall storage complexity is $\mathcal{O}(N)$.

The main computational complexity lies in the $\min\{k_{\text{ss}}, k_\epsilon\}$ matrix-vector multiplications that need to be performed (plus the subsequent multiplication

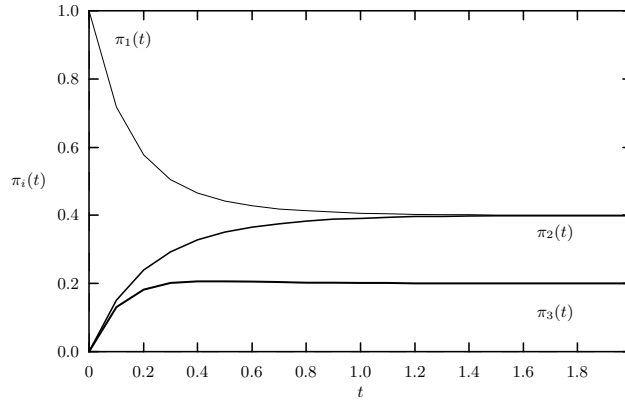


Fig. 15. First two seconds in the evolution of the three-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

of these vectors with the precomputed Poisson probabilities). As we have seen above, for large λt, k_ϵ is of order $\mathcal{O}(\lambda t)$. A single matrix-vector multiplication costs, in case of a sparse matrix $\mathbf{P}$ only $\mathcal{O}(N)$, and in case of a non-sparse matrix $\mathcal{O}(N^2)$. Taking the sparse case, we arrive at an overall time complexity of $\mathcal{O}(\lambda t N)$.

To increase the efficiency of uniformisation in specific situations, various variants have been developed. A good overview can be found in [53, 55, 54].

6.4 Uniformisation for Cumulative Measures

Let us now address a uniformisation-based efficient procedure for computing the expected accumulated reward over $[0, t)$, that is:

$$E[Y(t)] = E\left[\sum_{i=1}^{N} r_i l_i(t)\right]. \tag{61}$$

We first note that in the interval $[0, t)$, that is, an interval of length t, the expected time between two jumps, when k jumps have taken place according to a Poisson process with rate λ equals $t/(k+1)$. Interpreting λ as the uniformisation rate, the expected accumulated reward until time t, given k jumps, in the uniformised chain equals

$$\frac{t}{k+1} \sum_{i=1}^{N} r_i \sum_{m=0}^{k} \hat{\pi}_i(m).$$

This expression can be explained as follows. The right-most sum expresses the sum of the probabilities to reside in state i over the $k+1$ intervals addressed;

multiplied with the mean interval length (left-most factor), this gives the expected time spent in state i. The first summation weights these times with the corresponding reward and adds over all states.

We can now sum the above expression over all possible number of jumps that might occur during the interval $[0, t)$ and weight them with the usual Poisson probabilities, to arrive at:

$$E[Y(t)] = \sum_{k=0}^{\infty} \psi(\lambda t; k) \frac{t}{k+1} \sum_{i=1}^{N} r_i \sum_{m=0}^{k} \hat{\pi}_i(m). \tag{62}$$

Based on this expression, efficient numerical procedures can be devised as follows (see also [63,64]). First define

$$\phi(\lambda t; k) = \psi(\lambda t; k) \frac{t}{k+1} = e^{-\lambda t} \frac{\lambda^n t^{n+1}}{(k+1)!},$$

which can be computed recursively in a similar way as $\psi(\lambda t; k)$. When we define the diagonal matrix $\mathbf{R} = \text{diag}(\underline{r})$, i.e., $\mathbf{R}$ is a matrix with on the diagonal the rewards r_i, we can rewrite (62) by transforming the summation over all states in a matrix-vector multiplication as follows:

$$E[Y(t)] = \underline{1} \left(\sum_{k=0}^{\infty} \phi(\lambda t; k) \sum_{m=0}^{k} \underline{\hat{\pi}}(m) \right) \mathbf{R}, \tag{63}$$

where ϕ is computed recursively and $\underline{\hat{\pi}}(m) = \underline{\hat{\pi}}(m-1)\mathbf{P}$, with $\mathbf{P}$ the transition matrix for the uniformised DTMC. By defining the vector $\underline{C}(k)$, which denotes the cumulative probability over k steps to reside at each of the states, in the following way: $\underline{C}(0) = \underline{\hat{\pi}}(0)$ and $\underline{C}(k) = \underline{C}(k-1) + \underline{\hat{\pi}}(k)$, we finally arrive at

$$E[Y(t)] = \underline{1} \left(\sum_{k=0}^{\infty} \phi(\lambda t; k) \underline{C}(k) \right) \mathbf{R}. \tag{64}$$

As for the transient measures, a truncation criterion for the infinite summation can be easily developed. Similar storage and computational complexity considerations apply as in Section 6.3.

We finally comment on the solution of the performability distribution $F_Y(y, t)$, i.e., the probability distribution $\Pr\{Y(t) \leq y\}$ [50,51]. Also here, uniformisation can be employed; however, a direct summation over all states does not suffice any more. Instead, we have to sum the accumulated reward over all paths of length l (given a starting state) that can be taken through the DTMC, after which we have to compute a weighted sum over all these paths and their occurrence probabilities; for details we refer to [63,64,59].

7 Other Issues

In this section, a number of important issues not covered in detail in this chapter will be addressed briefly; pointers to relevant literature will be provided.

Phase-Type Distributions In this paper, we did not further address absorbing Markov chains, even though their applicability is substantial. Given a CTMC with a single absorbing state, the time from the initial state to absorption in that absorbing state has a so-called *phase-type distribution*, a distribution that can be seen as the sum of a possibly infinite number of exponential phases. Many well-known distributions are indeed of this type, e.g., the Erlang or an hyperexponential distribution. Moreover, almost any other distribution can be approximated very well with phase-type distributions. The birth-death processes we encountered in the examples, can be extended such that instead of exponential distributions, phase-type distributions are used. The thus resulting Markov chains are of so-called *quasi-birth-death* type and can still be solved efficiently using *matrix-geometric methods*, even when the state spece is infinitely large. For details, we refer to [58], [66, Chapter 5] or [27, Chapter 8].

Product-Form Solutions There is a large class of Markov chains that exhibits a so-called product-form solution. Most often such Markov chains arise when modelling systems not directly at the Markov chain level by identifying states and state-transitions, but when modelling systems as networks of queues. The structure of the Markov chain underlying the queueing network then results in an overall steady-state probability vector that can be written as the product of steady-state probabilities over smaller parts of the model. The book by Van Dijk on queueing networks and product-forms [18] is an excellent source on this topic. Also the more general books on performance evaluation mentioned above address product-form models. Hillston addresses product-form results for stochastic process algebras [35].

Distributed Solution of Markovian Models Especially when Markov chains are automatically generated from high-level specifications, these Markov chains tend to become very large. To cope with Markov chains with several millions (or more) states, specialised data structures have to be employed that are efficient both from a memory and a computational point of view. Recent advances in the use of tensor algebra and binary decision diagrams (and variants) should be mentioned here [15,13]. Furthermore, recently also the use of parallel and distributed computer systems has been advocated for both the generation of large Markov chains from high-level model specifications, as well as their numerical solution. Early work in this area can be found in [11,14]. With the PARSECS prototype tool, the generation and solution of Markov chains with more than 750 million states has recently been reported [7,28].

Tools for Markovian Modelling The practical application of Markovian modelling techniques has become widespread since the beginning of the 1980's. At that time, powerful workstations with larger memories became available for daily use. Since then, a large number of software tools has been built that support, in one way or another, the generation and solution of Markovian models.

Typically, the Markovian models are constructed using either general-purpose high-level modelling formalisms such as queueing networks (*cf.* QNAP2 [68], NUMAS [56] and MACOM [47]), stochastic Petri nets (*cf.* GreatSPN [12] and SPNP [16]), the "balls and buckets" formalism (*cf.* MARCA [65] and [66, Chapter 10.2–3]) stochastic activity networks (*cf.* UltraSAN [61,62]) or stochastic process algebras (*cf.* the PEPA workbench [34], TIPPtool [32] or TwoTowers [8]) or are more application-specific formalisms (*cf.* SAVE [23] for availability evaluation). It goes beyond the scope of the current paper to give an overview of all these tools; the interested reader is referred to a number of surveys: [31,30] and [29, Chapter 10].

Model Checking Markovian Models Recently, there has been an increased interest in the merging of Markovian modelling and evaluation techniques (as described in this paper) and techniques for formal system verification, in particular model checking [19,17,41]. Where previously timing aspects were not addressed in model checking, this becomes a necessity when model checking systems and protocols for real-time systems. By adding time in a specific stochastic manner to a finite-state machine, it can be interpreted as a Markov chain. By extending the logic to express time-related properties over the finite-state machine, as has been done with the logic CSL, a stochastically timed extension of CTL, such properties can be checked efficiently using evaluation techniques for Markov chains. Seminal work in this direction has been reported by Aziz *et al.* [2,1]; more recent developments can be found in [5,4,3,33].

8 Concluding Remarks

In the preceding sections we have addressed in a nutshell a large number of aspects of the use and solution of Markovian models. However, the amount of literature on Markovian models, their solution and application is vast. To conclude, let me refer to a number of well-known textbooks in the field. An absolute "must-read" on the numerical solution of Markov chains is Stewart's textbook [66]. Very readable is the two-volume work by Howard [37,38] as is the book by Kemeny and Snell [42]. A variety of books on performance evaluation in general address Markov chains in more or less detail, most often providing numerous examples [9,27,40,44,45,67].

References

1. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Lecture Notes in Computer Science 1102*, pages 269–276, 1996.
2. A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: the temporal logic of stochastic systems. In P. Wolper, editor, *Lecture Notes in Computer Science 939*, pages 155–165, 1995.

3. C. Baier, B.R. Haverkort, J.-P. Katoen, and H. Hermanns. On the logical characterisation of performability properties. In U. Montanari, J.D.P. Rolin, and E. Welzl, editors, *Lecture Notes in Computer Science 1853*, pages 780–792, 2000.
4. C. Baier, B.R. Haverkort, J.-P. Katoen, and H. Hermanns. Model checking continuous-time Markov chains by transient analysis. In E.A. Emerson and A.P. Sistla, editors, *Lecture Notes in Computer Science 1855*, pages 358–372, 2000.
5. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J.C.M. Baeten and S. Mauw, editors, *Lecture Notes in Computer Science 1664*, pages 146–161, 1999.
6. G. Balbo. Introduction to stochastic Petri nets. This volume.
7. A. Bell and B.R. Haverkort. Serial and parallel out-of-core solution of linear systems arising from generalised stochastic Petri net models. In *Proceedings High Performance Computing 2001*. Society for Computer Simulation, 2001.
8. M. Bernardo, R. Cleaveland, S. Sims, and W. Stewart. TwoTowers: a tool integrating functional and performance analysis of concurrent systems. In *Proceedings FORTE/PSTV 1998*, pages 457–467, 1998.
9. G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley & Sons, 1998.
10. E. Brinksma and H. Hermanns. Process algebra and Markov chains. This volume.
11. S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In G. De Michelis and M. Diaz, editors, *Applications and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 181–200. Springer-Verlag, 1995.
12. G. Chiola. A software package of the analysis of generalized stochastic Petri nets. In *Proceedings of the 1st International Workshop on Timed Petri Nets*, pages 136–143, Torino, Italy, July 1985. IEEE Computer Society Press.
13. G. Ciardo. Distributed and structured analysis approaches to study large and complex systems. This volume.
14. G. Ciardo, J. Gluckman, and D. Nicol. Distributed state space generation of discrete-state stochastic models. *INFORMS Journal of Computing*, 10(1):82–93, 1998.
15. G. Ciardo and A.S. Miner. Storage alternatives for large structured state spaces. In R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, editors, *Computer Performance Evaluation*, Lecture Notes in Computer Science 1245, pages 44–57. Springer Verlag, 1997.
16. G. Ciardo, J. Muppala, and K. S. Trivedi. SPNP: Stochastic Petri net package. In *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*, pages 142–151. IEEE Computer Society Press, 1989.
17. E.M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.
18. N.M. van Dijk. *Queueing Networks and Product Form: A Systems Approach*. John Wiley & Sons, 1993.
19. E.M.Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
20. A.K. Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *The Post Office Electrical Engineer's Journal*, 10:189–197, 1917.
21. B.L. Fox and P.W. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.
22. K.A. Frenkel. Allan L. Scherr — Big Blue's time-sharing pioneer. *Communications of the ACM*, 30(10):824–828, 1987.

23. A. Goyal, S.S. Lavenberg, and K.S. Trivedi. The system availability estimator. *Annals of Operations Research*, 8:285–306, 1987.
24. W.K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 357–371. Marcel Dekker, 1991.
25. D. Gross and D.R. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):343–361, 1984.
26. A.L. Hageman and D.M. Young. *Applied Iterative Methods*. Academic Press, 1981.
27. B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, 1998.
28. B. R. Haverkort, A. Bell, and H. Bohnenkamp. On the efficient sequential and distributed generation of very large Markov chains from stochastic Petri nets. In *Proceedings of the 8th International Workshop on Petri Nets and Performance Models*, pages 12–21. IEEE Computer Society Press, 1999.
29. B. R. Haverkort, R. Marie, G. Rubino, and K.S. Trivedi (editors). *Performability Modelling: Techniques and Tools*. John Wiley & Sons, 2001.
30. B.R. Haverkort and I.G. Niemegeers. Performability modelling tools and techniques. *Performance Evaluation*, 25:17–40, 1996.
31. B.R. Haverkort and K.S. Trivedi. Specification and generation of Markov reward models. *Discrete-Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.
32. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPTOOL. *Performance Evaluation*, 39:5–35, 2000.
33. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In S. Graf and M. Schwartzbach, editors, *Lecture Notes in Computer Science 1785*, pages 347–362. Springer-Verlag, 2000.
34. J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.
35. J. Hillston. Exploiting structure in solution: decomposing compositional models. This volume.
36. A.S. Hornby. *Oxford Advanced Learner's Dictionary of Current English*. Oxford University Press, 1974.
37. R.A. Howard. *Dynamic Probabilistic Systems; Volume I: Markov models*. John Wiley & Sons, 1971.
38. R.A. Howard. *Dynamic Probabilistic Systems; Volume II: Semi-Markov and decision processes*. John Wiley & Sons, 1971.
39. A. Jensen. Markov chains as an aid in the study of Markov processes. *Skand. Aktuarietidskrift*, 3:87–91, 1953.
40. K. Kant. *Introduction to Computer System Performance Evaluation*. McGraw-Hill, 1992.
41. J.-P. Katoen. *Concepts, Algorithms and Tools for Model Checking*. Universität Erlangen-Nürnberg, 1999.
42. J.G. Kemeny and J.L. Snell. *Finite Markov chains*. Van Nostrand, Princeton, 1960.
43. D.G. Kendall. Some problems in the theory of queues. *Journal of the Royal Statistical Society, Ser. B*, 13:151–185, 1951.
44. L. Kleinrock. *Queueing Systems; Volume 1: Theory*. John Wiley & Sons, 1975.
45. L. Kleinrock. *Queueing Systems; Volume 2: Computer Applications*. John Wiley & Sons, 1976.

46. A.N. Kolmogorov. Anfangsgründe der Theorie der Markoffschen Ketten mit unendlichen vielen möglichen Zuständen. *Mat. Sbornik N.S.*, pages 607–610, 1936.
47. U. Krieger, B. Müller-Clostermann, and M. Sczittnick. Modelling and analysis of communication systems based on computational methods for Markov chains. *IEEE Journal on Selected Areas in Communications*, 8(9):1630–1648, 1990.
48. V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, London, Glasgow, Weinheim, 1995.
49. A.A. Markov. Investigations of an important case of dependent trails. *Izvestia Acad. Nauk VI, Series I (in Russian)*, 61, 1907.
50. J.F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, 29(8):720–731, 1980.
51. J.F. Meyer. Closed-form solutions of performability. *IEEE Transactions on Computers*, 31(7):648–657, 1982.
52. C. Moler and C.F. van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–835, 1978.
53. A.P.A. van Moorsel. *Performability Evaluation Concepts and Techniques*. PhD thesis, University of Twente, 1993.
54. A.P.A. van Moorsel and B.R. Haverkort. Probabilistic evaluation for the analytical solution of large Markov models: Algorithms and tool support. *Microelectronics and Reliability*, 36(6):733–755, 1996.
55. A.P.A. van Moorsel and W.H. Sanders. Adaptive uniformization. *Stochastic Models*, 10(3):619–648, 1994.
56. B. Müller-Clostermann. NUMAS – a tool for the numerical analysis of computer systems. In D. Potier, editor, *Proceedings of the International Conference on Modelling Techniques and Tools for Performance Analysis*, pages 141–154. North-Holland, 1985.
57. J.K. Muppala and K.S. Trivedi. Numerical transient solution of finite Markovian queueing systems. In U. Bhat, editor, *Queueing and Related Models*. Oxford University Press, 1992.
58. M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Johns Hopkins University Press, 1981.
59. M.A. Qureshi and W.H. Sanders. A new methodology for calculating distributions of reward accumulated during a finite interval. In *Proceedings of the 26th Symposium on Fault-Tolerant Computer Systems (Sendai, Japan, June 1996)*, pages 116–125. IEEE Computer Society Press, 1996.
60. W.H. Sanders and J.-F Meyer. Stochastic activity networks: Formal definitions and concepts. This volume.
61. W.H. Sanders and J.F. Meyer. Reduced-base model construction for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.
62. W.H. Sanders, W.D. Obal, M.A. Qureshi, and F.K. Widnajarko. The UltraSAN modeling environment. *Performance Evaluation*, 24:89–115, 1995.
63. E. de Souza e Silva and H.R. Gail. Calculating availability and performability measures of repairable computer systems using randomization. *Journal of the ACM*, 36(1):171–193, 1989.
64. E. de Souza e Silva and H.R. Gail. Performability analysis of computer systems: from model specification to solution. *Performance Evaluation*, 1:157–196, 1992.
65. W.J. Stewart. MARCA: Markov chain analyzer. a software package for Markov modelling. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 37–62. Marcel Dekker, 1991.

66. W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.
67. K.S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. Prentice-Hall, 1982.
68. M. Veran and D. Potier. QNAP2: A portable environment for queueing system modelling. In D. Potier, editor, *Modelling Techniques and Tools for Computer Performance Evaluation*, pages 25–63. North-Holland, 1984.

Introduction to Stochastic Petri Nets

Gianfranco Balbo

Università di Torino, Torino, Italy,
Dipartimento di Informatica
balbo@di.unito.it

Abstract. Stochastic Petri Nets are a modelling formalism that can be conveniently used for the analysis of complex models of Discrete Event Dynami Systems (DEDS) and for their performance and reliability evaluation. The automatic construction of the probabilistic models that underly the dynamic behaviours of these nets rely on a set of results that derive from the theory of untimed Petri nets. The paper introduces the basic motivations for modelling DEDS and briefly overviews the basic results of net theory that are useful for the definition of Stochastic Petri Nets and Generalized Stochastic Petri Nets. The different approaches that have been used for introducing the concept of time in these models are discussed in order to provide the basis for the definition of SPNs and GSPNs as well. Details on the solution techniques and on ntheir computational aspects are provided. A brief overview of more advanced material is included at the end of the paper to highlight the state of the art in this field and to give pointers to relevant results published in the literature.

1 Introduction

Petri nets [60,1,59,63] are a powerful tool for the description and the analysis of systems that exhibit concurrency, synchronization and conflicts. Timed Petri nets [7,54] in which the basic model is augmented with time specifications are commonly used to evaluate the performance and reliability of complex systems.

The pioneering work in the area of timed Petri nets was performed by Merlin and Faber [50], and by Noe and Nutt [58]. In this early work, timed Petri nets were viewed as a formalism for the description of the global behaviour of complex structures. The nets were used to *tell* all the possible stories that systems could experience based on their temporal specifications and analysis was conducted on the basis of observations made on these stories.

Following these initial ideas, several proposals for incorporating timing information into Petri net models appeared in the literature. Interpreting Petri nets as state/event models, time is naturally associated with activities that induce state changes, and hence with the delays incurred before firing transitions.

Stochastic Petri Nets (SPNs) were introduced in 1980 [68,56,53] as a formalism for the description of Discrete Event Dynamic Systems (DEDS) whose

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 84–155, 2001.

dynamic behaviour could be represented by means of continuous-time homogeneous Markov chains. The original SPN proposal assumed atomic firings, exponentially distributed firing times, and a race execution policy; i.e., when multiple transitions are simultaneously enabled, the race policy selects the transition with the statistically minimum delay to fire.

With the aim of extending the modelling power of stochastic Petri nets, Generalized Stochastic Petri Nets (GSPNs) were proposed in [4]. GSPNs include two classes of transitions: exponentially distributed *timed* transitions, which are used to model the random delays associated with the execution of activities, and *immediate* transitions, which are devoted to the representation of logical actions that do not consume time. Immediate transitions permit the introduction of branching probabilities that are independent of timing specifications. When timed and immediate transitions are enabled in the same marking, immediate transitions always fire first. In GSPNs the reachability set is also partitioned in two sets. *Tangible* markings are those in which only timed transitions are enabled whereas *vanishing* markings are those in which at least one immediate transition is enabled. The time spent by a GSPN in a tangible state is exponentially distributed with the parameter depending on the timed transitions that are enabled in that marking; the time spent by a GSPN in a vanishing marking is instead zero. Other generalizations of the basic SPN formalism that are related to GSPNs, are the Extended Stochastic Petri Nets [35] and the Stochastic Activity Networks [51]. GSPNs are among the SPN formalisms that are most commonly used for the analysis of important problems and a considerable effort has been devoted to their improvement since the time of their original introduction.

In this paper, we discuss the relevance of this modelling formalism by providing first a broad view of its application field and by introducing the basic results that set the ground for the derivation of the stochastic processes corresponding to these models and for the study of their solution methods. The balance of the paper is the following. Section 2 briefly discusses the relevance of modelling to support the analysis and the design of complex systems. Section 3 introduces the characteristics of Discrete Event Dynamic Systems and discusses the role that models play in the analysis and the design of applications that can be represented within this framework. Section 4 describes the relevance of Petri nets for modelling Discrete Event Dynamic Systems and introduces the basic classical properties of the formalism that are needed later in the paper. Section 5 briefly surveys the impact that a global priority structure has on the properties and the behaviours of Petri nets. Section 6 presents the different possibilities that exist for introducing the concept of time in Petri net models. Section 7 introduces the definition of Stochastic Petri nets and provides the details for the construction of their underlying stochastic process. Section 8 discusses the characteristics of Generalized Stochastic Petri Nets and provides details on some of the computational issues that are relevant for the application of this modeling formalism. Section 9 concludes the paper with a few pointers to more advanced material. and with some general remarks on net modelling.

The paper has been written in the attempt of providing a uniform and self-contained introduction to the problem of modelling Discrete Event Dynamic Systems with Stochastic Petri Nets. The discussion introduces the basic terminology and operational rules of Petri nets for which a comprehensive reference can be found in [55] where the reader will be able to find a clear explanation of all the concepts that are only marginally addressed in this work.

2 Models

Understanding the behaviour of real systems is always difficult due to the complexity of their organization and to the intricacy of the interactions among their components.

Designing and managing real systems often require that relationships between organizational choices and attained results be identified in order to decide on possible improvements of the system architecture and of the operational environment for achieving better performance.

In all these cases, reasoning on the behaviour of systems can become more reliable if proper descriptions are available that help in clarifying the relationships among system components. The best way of obtaining such descriptions is that of constructing a model that highlights the important features of the system organization and provides ways of quantifying its properties neglecting all those details that are relevant for the actual implementation, but that are marginal for the objective of the study.

Models can be developed for a variety of reasons that include understanding and learning about the behaviour of the system, improving its performance and making decisions about its design or its operation. Models are useful for explaining why and how certain features of the system actually occur. The model helps in developing insights on the operation of the system and in understanding the directions of the influence that certain input parameters may have on the results.

The mathematical formulation of a model provides ways for formal reasoning about the behaviour of a real system in a manner that is safe (although difficult in same cases) and that is amenable for automatization.

The possibility of computing results from the analysis of a model is the key for closing a loop that starts from the abstraction of the relevant features of the system during modelling construction and that ends with the interpretation of the results provided by the model and reflected on the real system.

3 Discrete Event Dynamic Systems

A system is often defined as a collection of *objects* together with their *relations*. Objects are characterized by *attributes* some of which are fixed, while others are variable. The value assumed by a variable attribute contributes to the definition of the state of the object (*local* state). The *state* of the system (*global* state) results from the composition of the states of the individual objects (*components*). The

relations among objects represent the constraints that drive the change of states. Abstracting from the particular event that may cause the change of state, we call *transitions* such change of state patterns. We can say that state variables are *passive* elements, while transitions are *active*.

Discrete Event Dynamic Systems (DEDS) are systems with a discrete *state space* (i.e., they have a countable - possibly infinite - number of states) and whose evolution is not directly due to the passage of time, but to the occurrence of *events*. Depending on whether events happen at arbitrary points in time or only at precise instants, DEDS are called *asynchronous* or *synchronous*

3.1 DEDS as a View

Many real systems can be viewed as DEDS not because of their intrinsic characteristics, but because of the aspects of their behaviour that we want to emphasize. For instance, a water reservoir that contains a continuously varying quantity of water, can be viewed as a DEDS if we restrict our attention to the fact that the water exceeds or not a predefined safety level. In this case the reservoir can assume only two states (safe and un-safe) and the events that may cause the change of state can be identified in the occurrence of thunderstorms and in the openings of the dam.

We thus always speak of DEDS systems referring either to DEDS view of a real system or to a system that can be viewed in this way.

DEDS systems can be identified within quite different application domains. In Flexible Manufacturing, the state of the system may be represented by the number of parts present in front of the different machine-tools and the events may correspond to the completions of the activities performed by different machine-tools, to the production of a good, or to the arrival of new raw parts

In Computing, the state of the system may corresponds to the number of tasks currently in process as well as to those waiting for the completion of some I/O actions; examples of events are in this case the CPU quantum expiration, the interrupts coming from the I/O devices and the traps due to system call executions. In Telecommunication, the state may corresponds to the number of packets stored in the different buffers and the events to message submissions as well as to protocol actions. Finally, in Traffic the state of the system may corresponds to the number of cars waiting in a parking lot, or at a crossroad, or using a section of road while events may correspond to arrival and departures of cars as well as to changes of semaphore colours.

In all these cases, independently of the actual meaning of the different components, understanding the behaviour of these system is usually hard because of the intrinsic complexity of the problem (e.g., the number of machine tools and of parts of Flexible Manufacturing System may give rise to millions of states that may be difficult to envision and to keep under control), because of the subtle interference among components (e.g., in Traffic systems the existence of bottlenecks, temporarily hidden by the presence of other congestion points that generate huge traffic-jams, may defeat the expected improvements coming from the - often costly - removal of such delay causes), and of the paradoxical relationship

among the operations of certain components (e.g., in Time Sharing Computing System increasing the multiprogramming level to keep the CPU busy may yield an actual reduction of the CPU utilization because of the consequent increasing of the page fault rate - thrashing effect). Reasoning about the behaviour of such systems is difficult without the support of proper models.

Formal models are thus the basis for a better understanding of existent systems and for the effective and efficient design of new solutions.

3.2 Performance Evaluation of Discrete Event Dynamic Systems

One of the reasons for modeling DEDS is that of constructing a formal representation that can be used to drive design decisions toward efficient solutions and to optimize system operation to obtain the best results at the minimum cost.

As real systems may undergo failures, DEDS models must be capable of representing such failure/repair cycles in order to design real systems that perform well also in presence of temporary failures. Performance and Dependability Evaluation is the discipline that uses mathematical and simulation models for the computation of time-related performance indices such as resource utilization, system productivity and system response time accounting for system failures.

To compute these results the modelling formalism must include the possibility for time specifications (usually expressed in terms of the delay needed for performing a given action) and for routing/selection information.

The great diversity of real systems is commonly reflected at the level of time and selection specifications by means of random variables, thus leading to the construction and the solution of probabilistic models. Performance indices are evaluated for DEDS in these cases through the computation of the probability of finding the DEDS in each of its states.

4 Petri Net Models of DEDS

Petri nets (PNs) are abstract formal models that have been developed in search for natural, simple, and powerful methods for describing and analyzing the flow of information and control in systems. Petri nets have been originally proposed for the description and the analysis of systems in which concurrency and conflicts play a special role. In Petri nets, the state of the system derives from the combination of local state variables that allow a direct representation of concurrency, causality, and independence. Petri nets are graphically represented as collections of places and transitions connected by directed arcs so to form bi-partite graphs. The connections of transitions with places by means of input and output arcs represents the pre- and post-conditions, respectively, for the transitions to be enabled to fire. These conditions can be captured by the *incidence* matrix of the model that is the basis for the computation of a large set of *structural* results that represent the real advantage of using these type of models. The graphical aspect of these models are very attractive for practical modelling since they help in understanding how features of the real system are conveyed in the model.

In order to keep the PN models of DEDS concise, high level PN have been introduced that provide a form of abbreviation when repetition of similar subnets would make the model large and difficult to understand. Always to make models easier to understand, several extensions have been introduced in the basic PN formalism, often with the disadvantage of reducing the *analyzability* of the model (e.g., inhibitor arcs give to the Petri net formalism the computation power of Turing Machines, but their effect is, in general, neglected during the structural analysis of the net).

The behaviour of PNs is independent of time and environment and is characterized by the *non-deterministic* firing of transitions that are simultaneously enabled in a given marking. The connection of the formalism with reality is provided in this case by *interpretations* that incorporate in the model external constraints such as time considerations. Different extensions and different interpretations yield different PN based formalisms sharing some basic principles.

Petri nets are models consisting of two parts:

1. A net structure - an inscribed bipartite directed graph, that represents the static part of the system. The two types of nodes are called places and transitions and are represented as circles and boxes (or bars), respectively. Places correspond to state variables of the system and transitions to actions that induce changes of states. Arcs connecting places to transitions are called *input* arcs; *output* arcs connect instead transitions to places. Different types of inscriptions lead to various families of nets. When the inscriptions are natural numbers associated with arcs, named weights or multiplicities, Place/Transition (P/T) nets are obtained.
2. A marking - an assignment of tokens to places. The marking of a place represents its state value.

The specification of a PN model is completed by the definition of an initial marking. The dynamics of a system (i.e., its behaviour) is given by the evolution of the marking that is driven by few simple rules. The basic rule allows the occurrence of a transition when the input state values fulfill some conditions expressed by the arc inscriptions. The occurrence of a transition changes the values of its adjacent state variables (markings of input and output places) according to arc inscriptions again. This separation allows to reason on net based models at two different levels: *structural* and *behavioural*. From the former we may derive "fast" conclusions on the possible behaviours of the modelled system. Pure behavioural reasonings can be more conclusive, but they may require substantial computations, which in certain cases may not even be feasible. Structural reasoning may be regarded as an *abstraction* of the behavioural one: for instance, instead of studying whether a given system has a finite state space, we might address the problem of whether the state space is finite *for every* possible initial state; similarly, we could investigate whether *there exists* an initial marking that guarantees infinite activity, rather than verifying if this is the case for a given initial state.

A marked Petri net is formally defined by the following tuple

$$PN = (P,\ T,\ F,\ W,\ \boldsymbol{m}_0)$$

where
$P = (p_1,\ p_2,\ ...,\ p_P)$ is the set of places,
$T = (t_1,\ t_2,\ ...,\ t_T)$ is the set of transitions,
$F \subseteq (P \times T) \bigcup (T \times P)$ is the set of arcs,
$W : F \rightarrow \mathbb{N}$ is a weight function,
$\boldsymbol{m}_0 = (m_{01},\ m_{02},\ ...,\ m_{0P})$ is the initial marking.

As we have said, a marking $\boldsymbol{m}$ is an assignment of tokens to places and can thus be represented by a vector with as many components as there are places in the net: the $i-th$ component of such a vector represents the number of tokens assigned to place p_i. When nets are large, a more convenient notation for markings is that of expressing the assignment of tokens by means of a formal sum in which we explicitly represent the name of a marked place multiplied by the number of tokens assigned to it. A marking that has h tokens in place i and k tokens in place j (only) will be denoted as $\boldsymbol{m} = hp_i + kp_j$ (a formal sum denoting the multiset on P defined by the marking).

The dot notation is used for pre- and post-sets of nodes: ${}^\bullet v = \{u| < u, v > \in F\}$ and $v^\bullet = \{u| < v, u > \in F\}$. A pair comprising a place p and a transition t is called a *self-loop* if p is both input and output of t ($p \in {}^\bullet t \ \bigwedge \ p \in t^\bullet$ - see Figure 1).

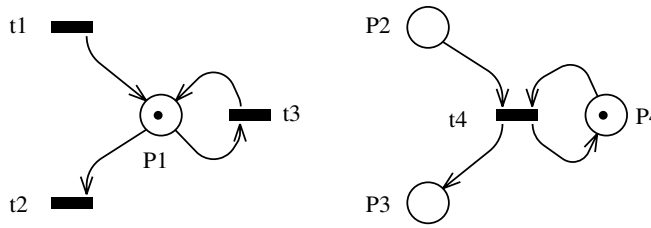


Fig. 1. Self-loops

A net is said to be pure if it has no self-loops. Pure nets are completely characterised by a single matrix $\boldsymbol{C}$ that is called the *incidence matrix* of the net and that combines the information provided by the flow relations and by the weight function.

$$\boldsymbol{C} = \begin{array}{c} \\ \text{places} \end{array} \overset{\overset{\textit{transitions}}{\downarrow}}{\boxed{c_{pt}}}$$

with $c_{pt} = c_{pt}^{+} + c_{pt}^{-} = w(t,p) - w(p,t)$

4.1 System Dynamics

The graph and matrix characterizations that we have described in the previous section represent the static component of a PN model. The dynamic evolution of the PN marking is governed by transition occurrences (firings) which consume and create tokens.

"Enabling" and "firing" rules are associated with transitions. The enabling rule states the conditions under which transitions are allowed to fire. The firing rule defines the marking modification induced by the occurrence of a transition. Informally, we can say that the enabling rule defines the *conditions* that allow a transition to fire, and the firing rule specifies the *change* of state produced by the transition.

Both the enabling and the firing rules are specified in terms of arc characteristics. In particular, the enabling rule involves (most of the time) input arcs only, while the firing rule depends on input and output arcs. Note that input arcs play a double role, since they are involved both in enabling and in firing.

A transition t is *enabled* if and only if each input place contains a number of tokens *greater or equal* than given thresholds defined by the multiplicities of arcs. Formally, this condition is expressed by the following definition:

Definition 1 (Enabling). *Transition t is enabled in marking $\boldsymbol{m}$ if and only if*

- $\forall p \in {}^{\bullet}t, m(p) \geq O(t,p)$
  that, in matrix notation is equivalent to
- $\boldsymbol{m} \geq \boldsymbol{c}(.,t)^{-T}$

The set of transitions enabled in marking $\boldsymbol{m}$ is indicated with $E(\boldsymbol{m})$; the number of simultaneous enablings of a transition t_i in a given marking $\boldsymbol{m}$ is called its enabling degree, and is denoted by $e_i(\boldsymbol{m})$.

When transition t fires, it *deletes* from each place in its input set ${}^{\bullet}t$ as many tokens as the multiplicity of the arc connecting that place to t, and *adds* to each place in its output set $t^{\bullet}$ as many tokens as the multiplicity of the arc connecting t to that place.

Definition 2 (Firing). *The firing of transition t, enabled in marking $\boldsymbol{m}$, produces marking $\boldsymbol{m}'$ such that*

- $\boldsymbol{m}' = \boldsymbol{m} + O(t) - I(t)$
  again, this same relation is expressed in matrix notation as follows
- $\boldsymbol{m}' = \boldsymbol{m} - \boldsymbol{c}(.,t)^{-T} + \boldsymbol{c}(.,t)^{+T}$

This statement is usually indicated in a compact way as $\boldsymbol{m}[t\rangle\boldsymbol{m}'$, and we say that $\boldsymbol{m}'$ is *directly reachable* from $\boldsymbol{m}$.

The natural extension of the concept of transition firing, is the firing of a *transition sequence* (or execution sequence). A transition sequence[1] $\sigma = t_{(1)}, \cdots, t_{(k)}$ can fire starting from marking $\boldsymbol{m}$ if and only if there exists a sequence of markings $\boldsymbol{m} = m_{(1)}, \cdots, m_{(k+1)} = \boldsymbol{m}'$ such that $\forall i = (1, \cdots, k), \boldsymbol{m}_{(i)}[t_{(i)}\rangle\boldsymbol{m}_{(i+1)}$. We denote by $\boldsymbol{m}[\sigma\rangle\boldsymbol{m}'$ the firing of a transition sequence, and we say that $\boldsymbol{m}'$ is *reachable* from $\boldsymbol{m}$.

An important final consideration is that the enabling and firing rules for a generic transition t are "local": indeed, only the numbers of tokens in the input of t, and the weights of the arcs connected to t need to be considered to establish whether t can fire and to compute the change of marking induced by the firing of t. This justifies the assertion that the PN marking is intrinsically "distributed".

A common way of describing the behaviour of a P/T system is by means of its sequential observation. A hypothetical observer is supposed to "see" only single events occurring at any point in time. The interleaving semantics of a net system is given by all possible sequences of individual transition firings that could be observed from the initial marking. If two transitions t_1 and t_2 are enabled simultaneously, and the occurrence of one does not disable the other, in principle they could occur at the same time, but the sequential observer will see either t_1 followed by t_2 or viceversa. The name interleaving semantics comes from this way of seing simultaneous occurrences.

Starting from the initial marking it is possible to compute the set of all markings reachable from it (the state space of the PN) and all the paths that the system may follow to move from state to state.[2]

Definition 3. *The Reachability Set of a PN with initial marking $\boldsymbol{m}_0$ is denoted $RS(\boldsymbol{m}_0)$, and is defined as the smallest set of markings such that*

- $\boldsymbol{m}_0 \in RS(\boldsymbol{m}_0)$
- $\boldsymbol{m}_1 \in RS(\boldsymbol{m}_0) \wedge \exists t \in T : \boldsymbol{m}_1[t\rangle\boldsymbol{m}_2 \Rightarrow \boldsymbol{m}_2 \in RS(\boldsymbol{m}_0)$

[1] We write $t_{(1)}$ rather than t_1 because we want to indicate the first transition in the sequence, that may not be the one named t_1.

[2] Obviously this computation is feasible only in the case of models with finite state spaces. In the rest of this paper, we assume that our models satisfy this condition, except when it is sted differently. Proper generalisations are possible to deal with infinite state spaces introducing the notion of "covering tree" [55].

When there is no possibility of confusion we indicate with RS the set $RS(\boldsymbol{m}_0)$. We also indicate with $RS(\boldsymbol{m})$ the set of markings reachable from a generic marking $\boldsymbol{m}$.

The RS contains no information about the transition sequences fired to reach each marking. This information is contained in the reachability graph, where each node represents a reachable state, and there is an arc from $\boldsymbol{m}_1$ to $\boldsymbol{m}_2$ if the marking $\boldsymbol{m}_2$ is directly reachable from $\boldsymbol{m}_1$. If $\boldsymbol{m}_1[t\rangle\boldsymbol{m}_2$, the arc is labelled with t. Note that more than one arc can connect two nodes (it is indeed possible for two transitions to be enabled in the same marking and to produce the same state change), so that the reachability graph is actually a multigraph.

Definition 4. *Given a PN system, and its reachability set RS, we call Reachability Graph $RG(\boldsymbol{m}_0)$ the labelled directed multigraph whose set of nodes is RS, and whose set of arcs A is defined as follows:*

$$\begin{array}{l} \bullet A \subseteq RS \times RS \times T \\ \bullet \langle \boldsymbol{m}_i, \boldsymbol{m}_j, t\rangle \in A \Leftrightarrow \boldsymbol{m}_i[t\rangle\boldsymbol{m}_j \end{array} \tag{1}$$

$\boldsymbol{m}_0$ is taken as the initial node of the graph.

Multiple events may happen at any given time. A *step* S is a multi-set of transitions that are enabled to concurrently fire in the same marking. Firing a step amounts to withdraw the tokens from all the input places of the transitions of the step and to deposit tokens in all their output places. If a set of transitions can be fired in a step, this makes explicit the fact that they need not occurring in a precise order. The occurrence of a step can be denoted by $\boldsymbol{m} \xrightarrow{S} \boldsymbol{m}'$, or $\boldsymbol{m} \xrightarrow{\sigma} \boldsymbol{m}'$, if σ is an arbitrary sequentialisation of S. In fact, every sequentialisation of the step is fireable so that, in practice, the reachable markings can be computed considering individual transition occurrences only.

The dynamic behaviour of Petri net models is characterized by three basic phenomena that account for the fact that actions may occur simultaneously (*concurrency*), some actions require that others occur first (*causal dependency*), and actions may occur only in alternative (*conflicts*).

Concurrency - Two transitions are concurrent in a given marking if they can occur in a step.

Definition 5. *Transitions t_i and t_j are in a concurrency relation in marking $\boldsymbol{m}$, denoted by $< t_i, t_j > \in CO(\boldsymbol{m})$, if $\boldsymbol{m} \xrightarrow{t_i} \boldsymbol{m}'$, $\boldsymbol{m} \xrightarrow{t_j} \boldsymbol{m}''$, $e_j(\boldsymbol{m}') > 0$, and $e_i(\boldsymbol{m}'') > 0$.*

in other words, $< t_i, t_j > \in CO(\boldsymbol{m})$, if $\boldsymbol{m} > \boldsymbol{c}(., t_i)^T + \boldsymbol{c}(., t_j)^T$. Notice that steps allow to express true concurrency. In the case of interleaving semantics, as we mentioned before, concurrency of two (or more) actions t_1 and t_2 is represented by the possibility of performing them in any order, first t_1 and then t_2, or viceversa. Nevertheless, the presence of all possible sequentialisations of

the actions does not imply that they are "truly" concurrent, as the example in Figure 2 illustrates: t_1 and t_2 can occur in any order, but they cannot occur simultaneously, and in fact the step $t_1 + t_2$ is not enabled. The distinction is especially important if transitions t_1 and t_2 were to be refined, i.e., if they were to be replaced by subnets.

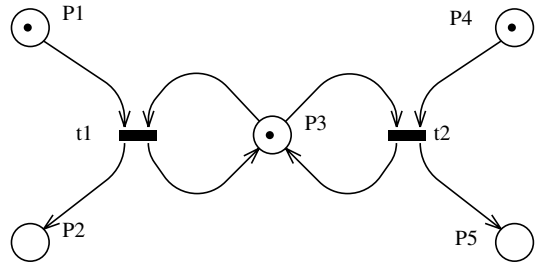


Fig. 2. Shared place

Causal Dependence - Informally, causal dependencies are represented by the partial ordering of actions induced by the flow relation. They correspond to situations in which the firing of a given transition can happen only after the occurrence of others in whatever order.

Definition 6. *Transitions t_i is in direct causality relation with t_j in marking $\boldsymbol{m}$, denoted by $< t_i, t_j > \in DC(\boldsymbol{m})$, if $\boldsymbol{m} \xrightarrow{t_i} \boldsymbol{m}'$, and $e_j(\boldsymbol{m}') > e_j(\boldsymbol{m})$.*

The very basic net construct used to model causal dependences is a place connecting two transitions. Transitions connected through a place are said to be in structural causal connection relation $< t_i, t_j > \in SCC(\boldsymbol{m})$, if $t_i^\bullet \bigwedge {}^\bullet t_j \neq \emptyset$.

Conflicts - Informally, we have a situation of *conflict* when, being several transitions enabled in the same marking, we have to chose which one to fire and, by so doing, we affect the enabling conditions of the others. We can thus say that a transition t_r is in conflict with transition t_s in marking $\boldsymbol{m}$ iff $t_r, t_s \in E(\boldsymbol{m})$, $\boldsymbol{m} \xrightarrow{t_s} \boldsymbol{m}'$, and $t_r \notin E(\boldsymbol{m})$. Things are more complex when we consider concurrent systems where the fact that two transitions are enabled in a given marking does not necessarily means that we have to choose which one to fire even if they share some input places. Formally, we have a situation of conflict when the set of transitions enabled in a given marking is not a step.

Definition 7. *Transition t_i is said to be in effective conflict relation with transition t_j in marking $\boldsymbol{m}$, denoted by $< t_i, t_j > \in EFC(\boldsymbol{m})$, if $\boldsymbol{m} \xrightarrow{t_i} \boldsymbol{m}'$, and $e_j(\boldsymbol{m}') < e_j(\boldsymbol{m})$.*

This relation is antisymmetric as we can see from the second example of Figure 3 where t'_2 is in effective conflict with t'_1, but not the other way around, since the firing of t'_1 does not decrease the enabling degree of t'_2.

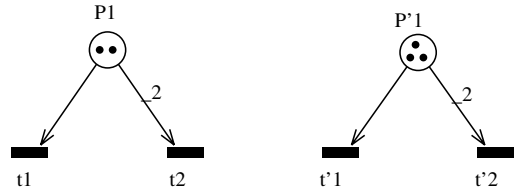


Fig. 3. Effective conflicts

The very basic net construct used to model conflicts is a place with more than one output transition. The output transitions of a place of this type are said to be in *structural conflict relation* ($< t_i, t_j > \in SC$, if $t_i^{\bullet} \bigwedge {}^{\bullet}t_j \neq \emptyset$.

This relation is reflexive and symmetric, but not transitive. Its transitive closure is named *coupled conflict relation* and partitions the transitions of a net into *coupled conflict sets*. *CCS(t)* denotes the coupled conflict set containing t. In Figure 4, transitions t_1 and t_2 are in structural conflict, while transitions t_3 and t_5 are not in structural conflict, but they are in coupled conflict relation, through t_4.

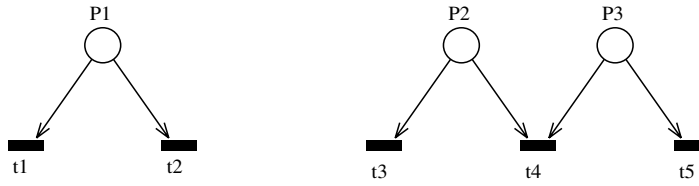


Fig. 4. Structural conflicts

It is important to remark the difference between structural conflicts and effective conflicts which depends on the marking of the net. A structural conflict makes possible the existence of an effective conflict, but does not guarantee it, except in the case of equal conflicts where all the transitions have the same input set.

Definition 8. *Transitions t_i and t_j are said to be in equal conflict relation, denoted by $< t_i, t_j > \in EQ$, if ${}^\bullet t_i = {}^\bullet t_j$*

Figure 5 shows an equal conflict set.

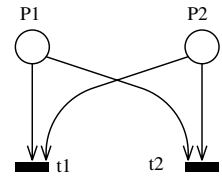


Fig. 5. Equal conflict

When structural conflicts are not equal, it may happen that transitions initially in conflict may subsequently become elements of a step or, viceversa, when the interleaved firing of a step yields to conflict situations. The cases depicted in Figure 6 show situations of this type.

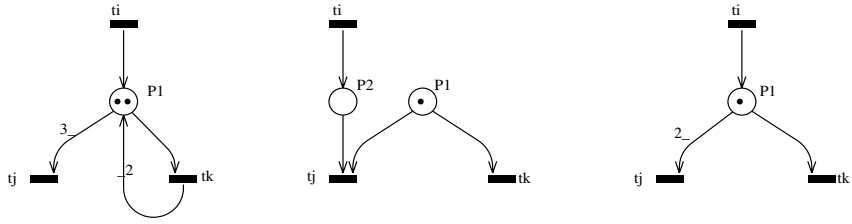


Fig. 6. Non equal conflicts

An intriguing situation arises when different interleaved firings of the members of a step may either yield conflict situations or not. This phenomenon is known as *confusion* and is again depicted by the conflicts of Figure 6, where we can recognize that t_i and t_k are a step such that, when t_k fires no effects are felt by transition t_i, while the same is not true in the other case.

Properties of Petri Nets - Properties of Petri net models are characteristics that allow to assess the quality of a given system in an objective manner. The following are among the most useful properties that can be defined for Petri net models.

Reachability and reversibility — As defined in the previous sections, a marking $\boldsymbol{m}'$ is *reachable* from $\boldsymbol{m}$ if there exists a sequence σ such that $\boldsymbol{m}[\sigma\rangle\boldsymbol{m}'$.

Reachability can be used to answer questions concerning the possibility for the modelled system of being in a given marking $\boldsymbol{m}$. An important reachability property is *reversibility*: a marked Petri net is said to be *reversible* if and only if from any state reachable from $\boldsymbol{m}_0$, it is possible to come back to $\boldsymbol{m}_0$ itself. More formally, a Petri net with initial marking $\boldsymbol{m}_0$ is reversible if and only if $\forall \boldsymbol{m} \in RS(\boldsymbol{m}_0), \boldsymbol{m}_0 \in RS(\boldsymbol{m})$. Reversibility expresses the possibility for a PN to come back infinitely often to its initial marking. In general, we say that $\boldsymbol{m} \in RS(\boldsymbol{m}_0)$ is a *home state* for the PN if and only if $\forall \boldsymbol{m}' \in RS(\boldsymbol{m}_0), \boldsymbol{m} \in RS(\boldsymbol{m}')$. A marking $\boldsymbol{m}_h$ is called a *home-state* iff

$$\forall \boldsymbol{m} \ \in \ RS(\boldsymbol{m}_0), \quad \boldsymbol{m}_h \ \in \ RS(\boldsymbol{m})$$

The set of the home-states of a Petri net is called its *home-space*

A Petri net is *reversible* whenever its initial marking $\boldsymbol{m}_0$ is a home-state

Liveness — A transition t is said to be *live* if and only if, for each marking $\boldsymbol{m}$ reachable from $\boldsymbol{m}_0$, there exists a marking $\boldsymbol{m}'$, reachable from $\boldsymbol{m}$, such that $t \in E(\boldsymbol{m}')$. Formally, transition t_r is *live* iff

$$\forall \boldsymbol{m} \ \in \ RS(\boldsymbol{m}_0), \quad \exists \boldsymbol{m}' \ : \ (\ \boldsymbol{m} \ \xrightarrow{s} \ \boldsymbol{m}' \bigwedge t_r \ \in \ E(\boldsymbol{m}') \ )$$

A Petri net is said to be live iff $\forall t_r \ \in \ T \ : \ t_r$ is live. Liveness is a property that depends on the initial marking. A transition that is not live is said to be *dead*. For each dead transition t, it is possible to find a marking $\boldsymbol{m}$ such that none of the markings in $RS(\boldsymbol{m})$ enables t.

A very important consequence of liveness is that, if at least one transition is live, then the Petri net cannot deadlock.[3] Moreover, if all transitions are live, then the corresponding Petri net contains no livelock.[4] Liveness defines the possibility for a transition to be enabled (and to fire) infinitely often.

Boundedness — A place p of a Petri net is said to be *k-bounded* if and only if, for each reachable marking $\boldsymbol{m}$, the number of tokens in that place is less than or equal to k. Formally, we have that a place p_i is bounded (k-bounded) iff

$$\forall \boldsymbol{m} \ \in \ RS(\boldsymbol{m}_0), \quad \exists k \ : \ m_i \leq \ k$$

A Petri net is said to be *k-bounded* if and only if all places $p \in P$ are *k-bounded*. Petri nets that are 1-bounded are said to be *safe*. A very important consequence of boundedness is that it implies the finiteness of the state space. In particular, if a Petri net comprising N places is *k-bounded*, the number of states cannot exceed $(k+1)^N$.

[3] A Petri net contains a deadlock if it can reach a state in which no transition can be fired.

[4] A system is in a livelock condition when it enters a subset of its activities from which it has no possibility of exiting.

It is interesting to note that boundedness, liveness and reversibility are (good) independent properties of a Petri net [55].

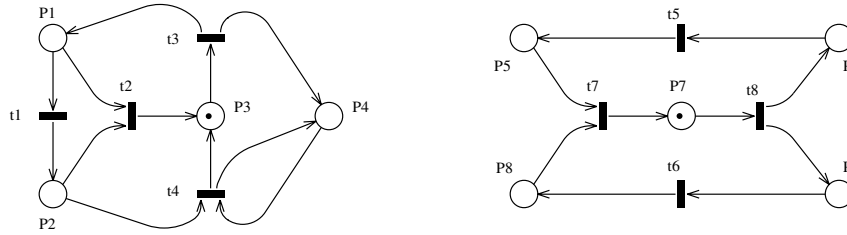


Fig. 7. Examples of two nets that are $\overline{BLR}$ and BLR, respectively.

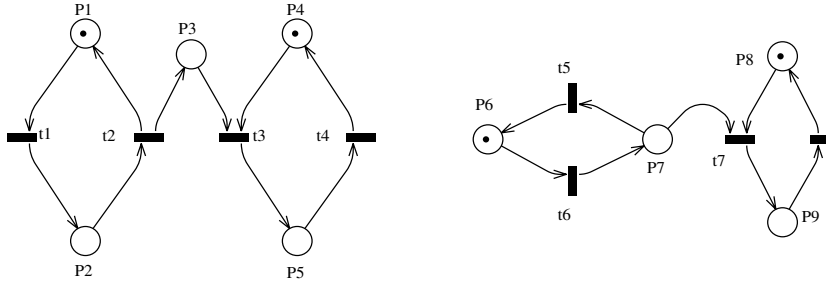


Fig. 8. Examples of two nets that are $\overline{B}LR$ and $B\overline{LR}$, respectively.

Mutual exclusion — Two mutual exclusion properties are of interest: one among places and one among transitions. Two places p and q are mutually exclusive in a Petri net if their token counts cannot be both positive in the same marking, i.e., $\forall \boldsymbol{m} \in RS \;\; \boldsymbol{m}(p) \cdot \boldsymbol{m}(q) = 0$. Two transitions in a PN are mutually exclusive if they cannot be both enabled in any marking.

Analysis Techniques - Depending on the techniques used for deriving these properties, they can be classified in the following way:

– *Structural* properties of Petri nets are obtained from the incidence matrix and from the graph structure of the model, independently of the initial marking.

– *Behavioural* properties of Petri nets depend on the initial marking and are obtained from the reachability graph (finite case) of the net or from the coverability tree (infinite case)[5].

Structural properties are quite interesting since they are proved directly from the structure of the model and are thus valid for every possible initial marking.

Linear Algebraic Techniques - Linear algebraic techniques, derive some basic properties of the net from the incidence matrix $\boldsymbol{C}$.

The relevance of the incidence matrix is due to the fact that it allows the net dynamics to be expressed by means of linear algebraic equations. In particular, we can observe that *for any marking* $\boldsymbol{m}$, the firing of a transition t enabled in $\boldsymbol{m}$ produces the new marking

$$\boldsymbol{m}' = \boldsymbol{m} + \boldsymbol{c}(., t)^{\mathrm{T}} \tag{2}$$

where $\boldsymbol{m}$ and $\boldsymbol{m}'$ are row vectors, and $\boldsymbol{c}(., t)$ is the column vector of $\boldsymbol{C}$ corresponding to transition t.

A similar relation holds for transition sequences. Given a transition sequence $\sigma = t_{(1)}, \cdots, t_{(k)}$, we define the transition count vector $\boldsymbol{v}_\sigma$ whose $i-th$ entry indicates how many times transition t_i appears in the sequence σ. $\boldsymbol{v}_\sigma$ is a $|T|$-component column vector. The marking $\boldsymbol{m}"$ obtained by firing the transition sequence σ from marking $\boldsymbol{m}$ ($\boldsymbol{m}[\sigma\rangle\boldsymbol{m}"$) can be obtained using the following equation:

$$\boldsymbol{m}" = \boldsymbol{m} + [\boldsymbol{C}\boldsymbol{v}_\sigma]^{\mathrm{T}} \tag{3}$$

Observe that only the number of times a transition fires is important: the order in which transitions appear in σ is irrelevant. The order is important for the definition of the transition sequence, and for checking whether the sequence can be fired, but it plays no role in the computation of the marking reached by that sequence. This remark leads to important consequences related to the definition of invariant relations for PN models.

P-semiflows and P-invariant relations — A Petri net is *strictly conservative* (or strictly invariant) iff

$$\sum_{p=1}^{P} m_p = \sum_{p=1}^{P} m_{0p}, \qquad \forall \boldsymbol{m} \in RS(\boldsymbol{m}_0)$$

Let us define a $|P|$-component weight column vector $\boldsymbol{y} = [y_1, y_2, \cdots, y_{|P|}]^{\mathrm{T}}$, whose entries are natural numbers. Consider the scalar product between the row

[5] The coverability tree provides a finite representation for infinite reachability graphs based on partial information. Details on this structure and on the algorithms for its construction can be found in [59].

vector representing an arbitrary marking $\boldsymbol{m}'$, and $\boldsymbol{y}$ (denoted $\boldsymbol{m}' \cdot \boldsymbol{y}$). A Petri net is *conservative* (or P invariant) iff

$$\exists\ \boldsymbol{y}\ =\ (y_1,\ y_2,\ ...,\ y_P) > 0 \ \textit{such that}$$

$$\sum_{p=1}^{P} y_p m_p\ =\ \sum_{p=1}^{P} y_p m_{0p} \qquad \forall \boldsymbol{m} \in RS(\boldsymbol{m}_0)$$

If $\boldsymbol{m}[t\rangle \boldsymbol{m}'$, then using (2) we can rewrite $\boldsymbol{m}' \cdot \boldsymbol{y}$ as:

$$\boldsymbol{m}' \cdot \boldsymbol{y} = \boldsymbol{m} \cdot \boldsymbol{y} + \boldsymbol{c}(.,t)^{\mathrm{T}} \cdot \boldsymbol{y} \tag{4}$$

Obviously, if $\boldsymbol{c}(.,t)^{\mathrm{T}} \cdot \boldsymbol{y} = 0$, the weighted token count in the Petri net (using the entries of $\boldsymbol{y}$ as weights) is the same for $\boldsymbol{m}$ and $\boldsymbol{m}'$. This means that the weighted token count is *invariant* with respect to the firing of t. More generally, if

$$\boldsymbol{C}^{\mathrm{T}} \cdot \boldsymbol{y} = \boldsymbol{0} \tag{5}$$

i.e., vector $\boldsymbol{y}$ is an integer solution of the set of linear equations

$$\forall t \in T: \quad \boldsymbol{c}(.,t)^{\mathrm{T}} \cdot \boldsymbol{y} = \boldsymbol{0} \tag{6}$$

then, no matter what sequence of transitions fires, the weighted token count does not change, and remains the same for any marking reachable from any given initial marking $\boldsymbol{m}$. The positive vectors $\boldsymbol{y}$ that satisfy Equation (5) are called the *P-semiflows* of the Petri net. Note that P-semiflows are computed from the incidence matrix, and are thus independent of any notion of initial marking. Markings are only instrumental for the interpretation of P-semiflows.

If $\boldsymbol{y}$ is an arbitrary vector of natural numbers, it can be visualized as a bag of places in which p_i appears with multiplicity y_i. This leads to the expression

$$\forall t \in T: \quad \sum_{p_i \in P} C(p_i, t) \cdot y_i = 0 \tag{7}$$

which identifies an invariant relation, stating that the sum of tokens in all places, weighted by $\boldsymbol{y}$, is constant for any reachable marking, and equal to $\boldsymbol{m}_0 \cdot \boldsymbol{y}$, for any choice of the initial marking $\boldsymbol{m}_0$. This invariant relation is called a place invariant, or simply *P-invariant*.

As a consequence, if in a PN model all places are covered by P-semiflows[6], then for any reachable marking (and independently of the initial marking), the maximum number of tokens in any place is finite (since the initial marking is finite) and the net is said to be *structurally bounded*.

All P-semiflows of a PN can be obtained as linear combinations of the P-semiflows that are elements of a minimal set PS. See [45,49,9,10] for P-semiflows computation algorithms.

[6] A place p is covered by a P-semiflow if there is at least one vector $\boldsymbol{y}$ with a non null entry for p.

T-semiflows and T-invariant relations — As observed in Equation (3), if $\boldsymbol{v}_\sigma$ is a firing count vector of a transition sequence σ, then

$$\boldsymbol{m}' = \boldsymbol{m} + [\boldsymbol{C}\boldsymbol{v}_\sigma]^{\mathrm{T}} \tag{8}$$

Obviously, if $[\boldsymbol{C}\boldsymbol{v}_\sigma]^{\mathrm{T}} = 0$, we obtain that $\boldsymbol{m}' = \boldsymbol{m}$ and we can observe that the firing sequence σ brings the PN back to the same marking $\boldsymbol{m}$. The vectors $\boldsymbol{x}$, that are integer solutions of the matrix equation

$$\boldsymbol{C} \cdot \boldsymbol{x} = \boldsymbol{0} \tag{9}$$

are called T-semiflows of the net. This matrix equation is equivalent to the set of linear equations

$$\forall p \in P : \quad \boldsymbol{c}(p, .) \cdot \boldsymbol{x} = 0 \tag{10}$$

In general, the invariant relation (called transition invariant or *T-invariant*) produced by a T-semiflow is the following:

$$\forall p \in P : \quad \sum_{t \in T} C(p, t) \cdot x(t) = 0 \tag{11}$$

This invariant relation states that, by firing from marking $\boldsymbol{m}$ any transition sequence σ whose transition count vector is a T-semiflow, the marking obtained at the end of the transition sequence is equal to the starting one, provided that σ can actually be fired from marking $\boldsymbol{m}$ ($\boldsymbol{m}[\sigma\rangle\boldsymbol{m}$). A net covered by T semiflows may have home states. A net with home states is covered by T-semiflows.

Note again that the T-semiflows computation is independent of any notion of marking, so that T-semiflows are identical for all PN models with the same structure and different initial markings.

Observe the intrinsic difference between P- and T-semiflows. The fact that all places in a Petri net are covered by P-semiflows is a sufficient condition for boundedness, whereas the existence of T-semiflows is only a necessary condition for a PN model to be able to return to a starting state, because there is no guarantee that a transition sequence with transition count vector equal to the T-semiflow can actually be fired.

Like P-semiflows, all T-semiflows can be obtained as linear combinations of the elements of a minimal set TS.

5 Petri Nets with Priority

A marked Petri net with transition priorities, arc multiplicities, and inhibitor arcs can be formally defined by the following tuple:

$$PN \;=\; (P,\; T,\; \Pi(.),\; I(.),\; O(.),\; H(.),\; \boldsymbol{m}_0)$$

- P is a set of places,
- T is a set of transitions,
- $\boldsymbol{m}_0$ is an initial marking,
- $\Pi(.)$, $I(.)$, $O(.)$, $H(.)$ are four functions defined on T.

The priority function $\Pi(.)$ maps transitions into non-negative natural numbers representing their priority level. The input, output, and inhibition functions $I(.)$, $O(.)$, and $H(.)$ map transitions on "bags" of places. The former two are represented as directed arcs from places to transitions and viceversa; the inhibition function is represented by circle-headed arcs. When greater than one, the multiplicity is written as a number next to the corresponding arc.

The priority definition that we assume in this paper is global: the enabled transitions with a given priority k always fire before any other enabled transition with priority[7] $j < k$.

This kind of priority definition can be used for two different modelling purposes: (1) it allows the partition of the transition set into classes representing actions at different logical levels, e.g. actions that take time versus actions corresponding to logical choices that occur instantaneously; (2) it gives the possibility of specifying a deterministic conflict resolution criterion.

Enabling and firing — The firing rule in Petri nets with priority requires the following new definitions:

- a transition t_j is said to *have concession* in marking $\boldsymbol{m}$ if the numbers of tokens in its input and inhibitor places verify the usual enabling conditions for PN models without priority $(\boldsymbol{m} \geq I(t)) \bigwedge (\boldsymbol{m} < H(t))$;
- a transition t_j is said to *be enabled* in marking $\boldsymbol{m}$ if it has concession in the same marking, and if no transition $t_k \in T$ of priority $\pi_k > \pi_j$ exists that has concession in $\boldsymbol{m}$. As a consequence, two transitions may be simultaneously enabled in a given marking only if they have the same priority level;
- a transition t_j can *fire* only if it is enabled. The effect of transition firing is identical to the case of PN models without priority.

Note that the presence of priority only restricts the set of enabled transitions (and therefore the possibilities of firing) with respect to the same PN model without priority. This implies that some properties are not influenced by the addition of a priority structure, while others are changed in a well-determined way, as we shall see in a while.

5.1 Conflicts, Confusion, and Priority

The notions of conflict and confusion are modified when a priority structure is associated with transitions. It is thus very important to be able to clearly identify by inspection of the net structure the sets of potentially conflicting transitions.

[7] Without loss of generality, we also assume that all lower priority levels are not empty, i.e.:

$$\forall t_j \in T, \quad \pi_j > 0 \implies \exists t_k \in T \ : \ \pi_k = \pi_j - 1$$

Conflict — The notion of conflict is drastically influenced by the introduction of a priority structure in PN models. The definition of effective conflict has to be modified with respect to the new notion of concession. Instead, the definition of enabling degree given in Section 4 remains unchanged for PN models with priority. Observe that this implies that both, transitions that have concession and enabled transitions, have enabling degree greater than zero. Conflict resolution causes the enabling degree of some transition to be reduced, and this may happen both for transitions with concession and for enabled ones. Hence the definition of the effective conflict relation is modified as follows.

Definition 9. *Transition t_i is in* effective conflict *relation with transition t_j in marking $\boldsymbol{m}$, (t_i $EC(\boldsymbol{m})$ t_j) iff t_j has concession in $\boldsymbol{m}$, t_i is enabled in $\boldsymbol{m}$, and the enabling degree of t_j decreases after the firing of t_i.*

Observe that a necessary condition for the EC relation to hold is that $\pi_i \geq \pi_j$, otherwise t_i would not be enabled in $\boldsymbol{m}$.

The definition of different priority levels for transitions introduces a further complication, since it destroys the locality of conflicts typical of PN models without priority. This observation leads to the possibility of *indirect conflicts*.

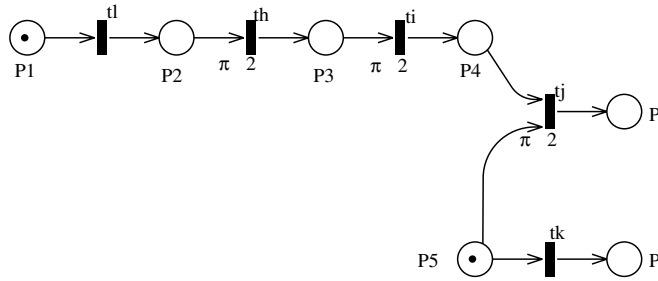


Fig. 9. An example of indirect conflict

Let us consider the net in Fig. 9. Transitions t_l and t_k are both enabled in the marking represented in the figure (since they both have concession, and no higher priority transition has concession), and apparently they are not in conflict, since they do not share input or inhibition places. According to the definition of concurrent transitions given in Section 4.1, one might conclude that t_l and t_k are concurrent. However, the firing of t_l enables a sequence of transitions t_h, t_i, and t_j which have higher priority than t_k, so that:

1. transition t_k becomes disabled while keeping its concession;
2. transition t_h is certainly the next transition to fire;
3. the firing of t_j removes the token from place p_5, thus taking concession away from transition t_k.

This sequence of events is not interruptible after the firing of t_l, due to the priority structure, and eventually results in the disabling of t_k through the firing of higher priority transitions. We call this situation *indirect effective conflict* between t_l and t_k.

Definition 10. *For any priority PN model* $\mathcal{M}_\pi$, $\forall t_i, t_j \in T$ *such that* $t_i \neq t_j$, $\forall \boldsymbol{m} : P \to \mathbb{N}$, *transition* t_i *is in indirect effective conflict with* t_j *in marking* $\boldsymbol{m}$ *(denoted* $t_i\ IEC(\boldsymbol{m})\ t_j$*) iff*

- t_j *has concession in* $\boldsymbol{m}$
- $t_i \in E(\boldsymbol{m})$
- $\exists \sigma = t_{(1)}, \ldots, t_{(k)}$ *such that*
  1. $\boldsymbol{m}[t_i\rangle \boldsymbol{m}_{(1)}[t_{(1)}\rangle \ldots \boldsymbol{m}_{(k)}[t_{(k)}\rangle \boldsymbol{m}'$, *and*
  2. $\forall 1 \leq h \leq k, \pi_{(h)} > \pi_j$, *and*
  3. $t_{(k)} EC(\boldsymbol{m}_{(k)}) t_j$.

Confusion and priority — In Section 4.1, the concept of confusion was discussed in the framework of PN models without priority. In this section we shall see how the introduction of a priority structure can avoid confusion.

Confusion is an important notion because it highlights the fact that, in terms of event ordering, the system behaviour is not completely defined: this underspecification could be due either to a precise modelling choice (the chosen abstraction level does not include any information on the ordering of events, hence the model analysis must investigate the effect of pursuing any possible ordering) or to a modelling error. The introduction of a priority structure may force a deterministic ordering of conflict resolutions that removes confusion.

For instance, let us consider again the example depicted in Fig. 9 assuming first that transitions t_l and t_k have the same priority level of the others. A confusion situation arises due to the fact that both sequences $\sigma = t_k, t_l, t_h, t_j$ and $\sigma' = t_l, t_h, t_i, t_k$ are fireable in marking $\boldsymbol{m} = p_1 + p_5$, and that they involve different conflict resolutions. By making the priority level of transitions t_h, t_i, and t_j higher than that of t_l and t_k we have removed the confusion situation since any conflict between t_j and t_k is always solved in favour of t_j; as a consequence the sequence $\sigma' = t_l, t_h, t_i, t_k$ is not fireable in $\boldsymbol{m}$ and confusion is avoided.

A structural necessary condition for indirect conflict is the presence of a non free-choice conflict comprising at least two transitions t_i and t_j at the same priority level and a third transition t_k causally connected to either t_i or t_j and such that $\pi_k = \pi_i = \pi_j$.

Structural conflict — For PN models without priority, we defined the notion of *structural conflict* relation (SC) to identify potentially conflicting pairs of transitions by inspection of the net structure. Intuitively, two transitions t_i and t_j are in structural conflict if they share at least one input place or if the output set of t_i is not disjoint from the inhibition set of t_j. This definition does not change for Petri nets with priority.

In this case we can also define the *indirect structural conflict* (ISC) relation that gives a necessary condition for two transitions to be in indirect effective

conflict relation in some marking. Intuitively, we have first to find a pair of transitions t_j and t_k such that $\pi_j > \pi_k$ and $t_j SC t_k$; then we have to follow the net arcs backwards starting from transition t_j until we find a new transition t_l such that $\pi_l \leq \pi_k$. All the transitions on the path (including t_l) with priority greater than or equal to π_k, are in indirect structural conflict relation with t_k. Indeed, any transition on the path can trigger a firing sequence of transitions with priority higher than π_k, that may eventually enable transition t_j whose firing would decrease the enabling degree of transition t_k. Notice that the transitions on the path are in causal connection relation. A formal recursive definition for the ISC relation follows:

Definition 11. *Given a priority PN model, two transitions t_l and t_k are in indirect structural conflict relation (denoted t_l ISC t_k) iff*

- $\pi_l \geq \pi_k$;
- $\exists t_j : \ (\pi_j > \pi_k) \wedge (t_k SCC t_j) \wedge ((t_j SC t_k) \vee (t_j ISC t_k))$

where SCC is the structural causal connection relation defined in the previous section.[8]

Let us consider the model of Fig. 9 again; since transition t_j is in SC relation with transition t_k, $\pi_j > \pi_k$ and $t_l SCC t_j$, it is possible to conclude that $t_l ISC t_k$.

Given the definition of structural conflict, it is possible to introduce the notion of *conflict set* and *extended conflict set* whose motivations will become apparent in the following sections.

We define the *symmetric structural conflict* as follows:

Definition 12. *Transition t_i is in* symmetric structural conflict *with t_j (denoted t_i SSC t_j) iff*

- $\pi_i = \pi_j$ *and*
- $t_i SC t_j \ \vee \ t_j SC t_i \ \vee \ t_i ISC t_j \ \vee \ t_j ISC t_i$.

The conflict set associated with a given transition t_i is the set of transitions that might be in conflict with t_i in some marking.

Definition 13. *The conflict set associated with a given transition t_i is defined as*

$$CS(t_i) = \{t_j : (t_i SSC t_j)\}$$

The transitive closure of the SSC relation is an equivalence relation that allows the partition of the set T into equivalence classes called *extended conflict sets*.

Definition 14. $ECS(t_i) = \{t_j : t_i \ SSC^* \ t_j \wedge \sim (t_i \ SME \ t_j)\}$.

[8] If the PN allows the presence of inhibitor arcs, the SCC relation must also account for the fact that t_k is causally connected with t_j also in case its firing decreases the marking of some of the places that are part of the (non-empty) inhibitor set of t_j.

In any marking that enables transitions of the same *ECS*, a choice that may have effect on the future evolution of the net must be made in order to decide which, among these transitions, has to be fired next

Two simultaneously enabled transitions t_i and t_j that belong to different *ECS* can be fired in any order.

5.2 Properties of Petri Nets with Priority

In order to briefly discuss the impact that priorities have on the properties of PN models, we must first divide them into two broad classes. Properties that hold for all states in the state space are called *safety* or *invariant* properties; properties that instead hold only for some state in the state space are called *eventuality* or *progress* properties). Examples of invariant properties are boundedness, and mutual exclusion. Examples of eventuality properties are reachability (a given marking will be eventually reached) and liveness (a transition will eventually become enabled).

Let $\mathcal{M}_\pi$ be a PN model with priority and let $\mathcal{M}$ be the underlying PN model without priority. Since the introduction of priority can only reduce the state space, all the safety properties that can be shown to be true for $\mathcal{M}$, surely hold also for $\mathcal{M}_\pi$. Eventuality properties instead are not preserved in general by the introduction of a priority structure.

It is interesting to observe that P and T-invariants describe properties that continue to hold after the addition of a priority structure. The reason is that they are computed only by taking into account the state change caused by transition firing, without any assumption on the possibility for a transition of ever becoming enabled. Boundedness is preserved by the introduction of a priority structure in the sense that a bounded PN model remains bounded after the introduction of a priority specification. This implies that the use of P-semiflows to study the boundedness of a PN model can be applied to the model without priority $\mathcal{M}$ associated with a priority PN model $\mathcal{M}_\pi$ and if the former model is shown to be structurally bounded, the conclusion can be extended to the latter model. Observe, however, that an unbounded PN model may become bounded after the specification of an appropriate priority structure.

On the other hand, since enabling is more restricted than in the corresponding PN model without priority, reachability is not preserved in general by the addition of a priority structure. However, a marking $\boldsymbol{m}'$ is reachable from a marking $\boldsymbol{m}$ in a PN model with priority only if it is reachable in the corresponding PN model without priority.

Liveness is intimately related to the enabling and firing rules, hence it is greatly influenced by a change in the priority specification: a live PN model may become not live after the introduction of an inappropriate priority structure and, viceversa, a PN model that is not live, may become live after the addition of an appropriate priority structure.

Expressive Power - According to their definition, P/T nets do not allow modelling zero tests, i.e., transitions that are enabled only if some place is empty.

The introduction of priorities and inhibitor arcs yields such a feature, making the extended model less amenable for analysis as we have just seen during the discussion of the properties of Petri nets with priorities. On the other hand, with the addition of inhibitor arcs and priorities, Petri nets increase their modelling power, actually leading to Turing machines [59].

It is possible to implement a system with inhibitor arcs using priorities and viceversa, even in the unbounded case, while preserving concurrent semantics, so both extensions can be interchanged except for modelling convenience (the transformations are rather cumbersome [23]). Inhibitor arcs have the advantage that they are graphically represented in the net structure, while the influence of a priority definition on the enabling of some transition is not so clearly reflected and is not so local. On the other hand, priorities arise naturally when a timing interpretation is considered. Therefore, despite their formal equivalence, both extensions are allowed on equal footing, because they have been introduced to cope with different situations. Figure 10 shows an example in which the repeated firing of t_1 accumulates tokens in P_1 until the operation mode switches and all the accumulated tokens are consumed (repeated firing of t_2). When P_1 becomes empty t_3 fires and the system goes back in the initial state. The representations with inhibitor arcs and priorities yield exactly the same behaviour.

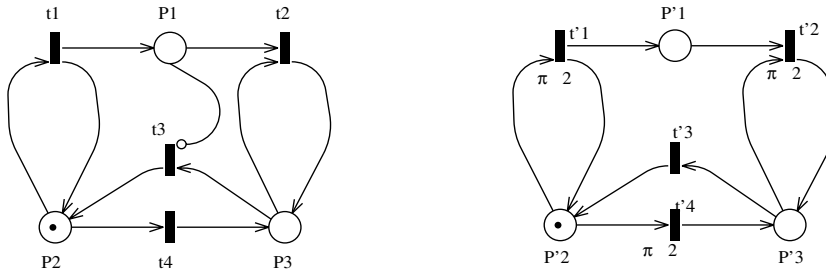


Fig. 10. Equivalence between inhibitor arcs and priority specifications

Inhibitor arcs deriving from bounded places can be implemented with the use of multiplicity and complementary places (preserving the interleaving semantics), as shown in the example of Fig. 11, which is equivalent to that of Fig. 10 assuming that place P_1 cannot contain more than 10 tokens. This simple transformation does not work so well, however, when concurrent semantics is considered (see [12] for details).

6 Time in Petri Nets

In this section we discuss the issues related to the introduction of *temporal concepts* into PN models. Particular attention will be given to the temporal seman-

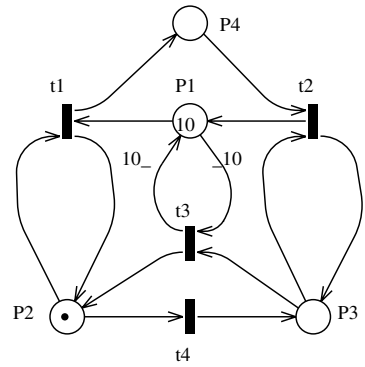


Fig. 11. Representations of inhibitor arcs with complementary places

tics that is peculiar to stochastic PNs (SPNs) and generalized SPNs (GSPNs). For this reason we shall always assume that timed transitions are associated with temporal specifications such that the simultaneous firing of two or more timed transitions can be neglected (this event has probability zero in SPNs and GSPNs).

6.1 The Motivations for Timing

The PN models that were considered in the previous sections included no notion of time. The concept of time was intentionally avoided in the original work by C.A.Petri [60], because of the effect that timing may have on the behaviour of PNs. In fact, the association of timing constraints with the activities represented in PN models may prevent certain transitions from firing, thus destroying the important assumption that all possible behaviours of a real system are represented by the structure of the PN.

In [59], the first book on PNs that dealt extensively with applications, the only remark about timed PNs was the following: "*The addition of timing information might provide a powerful new feature for PNs, but may not be possible in a manner consistent with the basic philosophy of PNs*". This attitude towards timing in PN models is due to the fact that PNs were originally considered as formal automata and investigated in their theoretical properties. Most of the early questions raised by researchers thus looked into the fundamental properties of PN models, into their analysis techniques and the associated computational complexity, and into the equivalence between PNs and other models of parallel computation. When dealing with these problems, timing is indeed not relevant.

Very soon PNs were however recognized as possible models of real concurrent systems, capable of coping with all aspects of parallelism and conflict in asynchronous activities with multiple actors. In this case, timing is not important when considering only the logical relationships between the entities that are part

of the real system. The concept of time becomes instead of paramount importance when the interest is driven by real applications whose efficiency is always a relevant design problem. Indeed, in areas like hardware and computer architecture design, communication protocols, and software system analysis, timing is crucial even to define the logical aspects of the dynamic operations.

Time is introduced in Petri nets to model the interaction among several activities considering their starting and completion instants The introduction of time specifications corresponds to an interpretation of the model by means of the observation of the autonomous (untimed) model and the definition of a non-autonomous model.

The pioneering works in the area of timed PNs were performed by P.M.Merlin and D.J.Farber [50], and by J.D.Noe and G.J.Nutt [58]. In both cases, PNs were not viewed as a formalism to statically model the logical relationships among the various entities that form a real system, but as a tool for the description of the global behaviour of complex structures. PNs were used to *tell* all the possible stories that the system can experience, and the temporal specifications were an essential part of the picture.

When introducing time into PN models, it would be extremely useful not to modify the basic behaviour of the underlying untimed model. By so doing, it is possible to study the timed PNs exploiting the properties of the basic model as well as the available theoretical results. The addition of temporal specifications therefore must not modify the unique and original way of expressing synchronization and parallelism that is peculiar to PNs. This requirement obviously conflicts with the user's wishes for extensions of the basic PN formalism to allow a direct and easy representation of specific phenomena of interest. Time specifications are also used to provide ways of reducing the non-determinism of the model by means of rules based on time considerations. Finally, time extensions must provide methods for the computation of performance indices.

Different ways of incorporating timing information into PN models have been proposed by many researchers during the last two decades; the different proposals are strongly influenced by the specific application fields and can be summarized as follows:

- **Timed places** - time may be associated with *places*:
  - tokens generated in an output place become available to fire a transition only after a delay has elapsed; the delay is an attribute of the place.
- **Timed tokens**- time may be associated with *tokens*:
  - tokens carry a time-stamp that indicates when they are available to fire a transition; this time-stamp can be incremented at each transition firing.
- **Timed arcs** - time may be associated with *arcs*:
  - a travelling delay is associated with each arc; tokens are available for firing only when they reach a transition.
- **Timed transitions** - time may be associated with *transitions*:
  - activity start corresponds to transition enabling,
  - activity end corresponds to transition firing.

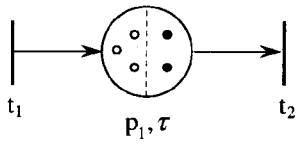


Fig. 12. Timed places

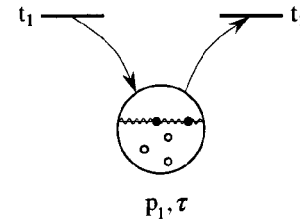


Fig. 13. Timed tokens

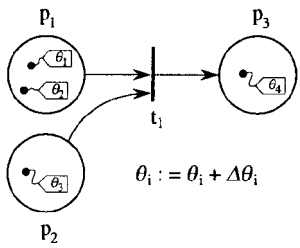


Fig. 14. Timed arcs

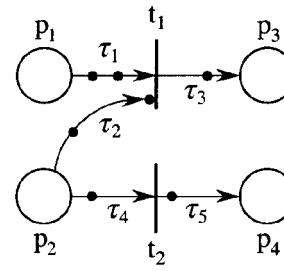


Fig. 15. Timed transitions

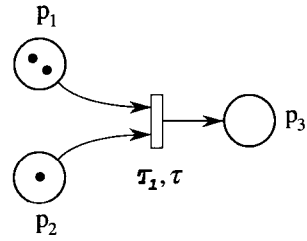
Fig.s 12, 13, 14, and 15, depicts in a compact manner these different possibilities with the purpose of helping the reader in grasping the differences between the various extensions that are sometimes subtle and difficult to identify at first glance. The reader interested in understanding better the implications of the different proposals is referred to the original papers [62,71,44,61,50,65,32,70].

6.2 Timed Transitions

Timed transitions represent the most common extension used by the authors to add time to PN models. The firing of a transition in a PN model corresponds to the event that changes the state of the real system. This change of state can be due to one of two reasons: it may either result from the verification of some logical condition in the system, or be induced by the completion of some activity. Considering the second case, we note that transitions can be used to model activities, so that transition enabling periods correspond to activity executions and transition firings correspond to activity completions. Hence, time can be naturally associated with transitions.

Different firing policies may be assumed: the *three-phase firing* assumes that tokens are consumed from input places when the transition is enabled, then the delay elapses, finally tokens are generated in output places; *atomic firing* assumes instead that tokens remain in input places during the whole transition

delay; they are consumed from input places and generated in output places when the transition fires.



Timed transition Petri nets (TTPN) with atomic firing can preserve the basic behaviour of the underlying untimed model. It is thus possible to qualitatively study TTPN with atomic firing exploiting the theory developed for untimed (autonomous) PN (reachability set, invariants, etc.). Timing specifications may affect the qualitative behaviour of the PN only when they describe *constant* and *interval* firing delays.

We can explain the behaviour of a timed transition (whose graphical representation is usually a box or a thick bar and whose name usually starts with T) by assuming that it incorporates a timer. When the transition is enabled, its local clock is set to an initial value. The timer is then decremented at constant speed, and the transition fires when the timer reaches the value zero. The timer associated with the transition can thus be used to model the duration of an activity whose completion induces the state change that is represented by the change of marking produced by the firing of T. The type of activity associated with the transition, whose duration is measured by the timer, depends on the DEDS that we are modelling: it may correspond to the execution of a task by a processor, or to the transmission of a message in a communication network, or to the work performed on a part by a machine tool in a manufacturing system. It is important to note that the activity is assumed to be in progress while the transition is enabled. This means that in the evolution of more complex nets, an interruption of the activity may take place if the transition loses its enabling condition before it can actually fire. The activity may be resumed later on, during the evolution of the net in the case of a new enabling of the associated transition. This may happen several times until the timer goes down to zero and the transition finally fires.

It is possible to define a *timed transition sequence* or *timed execution* of a timed PN system as a transition sequence (as defined in Section 4.1) augmented with a set of nondecreasing real values describing the epochs of firing of each transition. Such a timed transition sequence is denoted as follows:

$$[(\tau_{(1)}, T_{(1)}); \cdots; (\tau_{(j)}, T_{(j)}); \cdots]$$

The time intervals $[\tau_{(i)}, \tau_{(i+1)})$ between consecutive epochs represent the periods during which the PN sojourns in marking $\boldsymbol{m}_{(i)}$. This sojourn time corresponds

to a period in which the execution of one or more activities is in progress and the state of the system does not change.

6.3 Immediate Transitions

As we noted before, not all the events that occur in a DEDS model correspond to the end of time-consuming activities (or to activities that are considered time-consuming at the level of detail at which the model is developed). For instance, a model of a multiprocessor system described at a high level of abstraction often neglects the durations of task switchings, since these operations require a very small amount of time, compared with the durations of task executions. The same can be true for bus arbitration compared with bus use. In other cases, the state change induced by a specific event may be quite complex, and thus difficult to obtain with the firing of a single transition. Moreover, the state change can depend on the present state in a complex manner. As a result, the correct evolution of the timed PN model can often be conveniently described with subnets of transitions that consume no time and describe the logics or the algorithm of state evolution induced by the complex event.

To cope with both these situations in timed PN models, it is convenient to introduce a second type of transition called *immediate*. Immediate transitions fire as soon as they become enabled (with a null delay), thus acquiring a sort of precedence over timed transitions. In this paper, immediate transitions are depicted as thin bars whereas timed transitions are depicted as boxes or thick bars.

6.4 Parallelism and Conflict

The introduction of temporal specifications in PN models must not reduce the modelling capabilities with respect to the untimed case. Let us verify this condition as far as parallelism and conflict resolution are considered.

Pure parallelism can be modelled by two transitions that are independently enabled in the same marking. The evolution of the two activities is measured by the decrement of the clocks associated with the two transitions. When one of the timers reaches zero, the transition fires and a new marking is produced. In the new marking, the other transition is still enabled and its timer can either be reset or not depending on the different ways of managing this timer that will be discussed in the next section.

Consider now transitions T_1 and T_2 in Fig. 16. In this case, the two transitions are in free-choice conflict. In untimed PN systems, the choice of which of the two transitions to fire is completely nondeterministic. When more than one timed transition with atomic firing is enabled, the behaviour is similar, but a problem arises: *Which one of the enabled transitions is going to fire?* Two alternative selection rules are possible:

- **preselection** - the enabled transition that will fire is chosen when the marking is entered, according to some metric (e.g., priority),

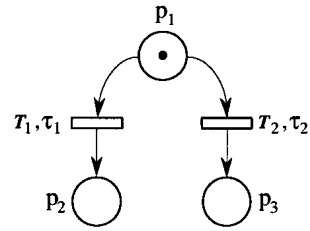


Fig. 16. Conflicting transitions

– **race**- the enabled transition that will fire is the one whose firing delay is minimum.

In the case of timed PNs, the conflict resolution depends on the delays associated with transitions and is obtained through the so-called *race* policy: when several timed transitions are enabled in a given marking $\boldsymbol{m}$, the transition with the shortest associated delay fires first (thus disabling the possible conflicting transitions).

Having discussed the conflict resolution policy among timed transitions, it is important to emphasize the fact that, when two or more immediate transitions are enabled in the same marking, some rule must be specified to select the one to fire first, thus ordering the firing of immediate transitions. Two types of rules will be used when situations of this type occur in the following. The first one is based on a deterministic choice of the transition to fire using the mechanism of *priority*. A second mechanism consists in the association of a discrete probability distribution function with the set of conflicting transitions. In this case the conflicts among immediate transitions are randomly solved.

In some cases, however, it may be desirable to separate conflict resolution from timing specification of transitions. Immediate transitions can be used to obtain this separation. The conflict can be transferred to a barrier of conflicting immediate transitions, followed by a set of timed transitions. The extensive use of this technique can eliminate from a net all conflicts among timed transitions that are simultaneously enabled in a given marking. If this mechanism is consistently used to prevent timed transitions from entering into conflict situations, a *preselection* policy of the (timed) transition to fire next is said to be used.

Conflicts comprising timed and immediate transitions have an important use in timed PNs: they allow the interruption (or preemption) of ongoing activities, when some special situation occurs. Consider, for example, the subnet in Fig. 17. A token in place p_1 starts the activity modelled by timed transition T_1. If a token arrives in p_2 before the firing of T_1, immediate transition t_2 becomes enabled and fires, thus disabling timed transition T_1. This behaviour again provides an example of the precedence of immediate over timed transitions.

The presence of immediate transitions induces a distinction among markings. Markings in which no immediate transitions are enabled are called *tangible*, whereas markings enabling at least one immediate transition are said to be

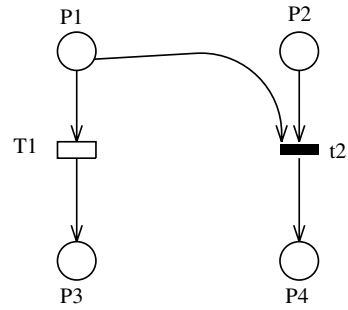


Fig. 17. Interrupting an activity with an immediate transition

vanishing. The timed PN system spends a positive amount of time in tangible markings, and a null time in vanishing markings.

6.5 Memory

An important issue that arises at every transition firing, when timed transitions are used in a model, is how to manage the timers of all the transitions that do not fire.

From the modeling point of view, the different policies that can be adopted link the past history of the systems to its future evolution considering various ways of retaining *memory* of the time already spent in activities. The question concerns the *memory policy* of transitions, and defines how to set the transition timers when a state change occurs, possibly modifying the enabling of transitions. Two basic mechanisms can be considered for a timed transition at each state change.

- **Continue**. The timer associated with the transition holds the present value and will *continue* later on the count-down.
- **Restart**. The timer associated with the transition is *restarted*, i.e., its present value is discarded and a new value will be generated when needed.

To model the different behaviours arising in real systems, different ways of keeping track of the past are possible by associating different continue or restart mechanisms with timed transitions. We discuss here three alternatives:

- **Resampling**. At each and every transition firing, the timers of all the timed transitions are discarded (restart mechanism). No memory of the past is recorded. After discarding all the timers, new values of the timers are set for the transitions that are enabled in the new marking.
- **Enabling memory**. At each transition firing, the timers of all the timed transitions that are disabled are restarted whereas the timers of all the timed

transitions that are not disabled hold their present value (continue mechanism). The memory of the past is recorded with an *enabling memory variable* associated with each transition. The enabling memory variable accounts for the work performed by the activity associated with the transition since the last instant of time its timer was set. In other words, the enabling memory variable measures the enabling time of the transition since the last instant of time it became enabled.

– **Age memory**. At each transition firing, the timers of all the timed transitions hold their present values (continue mechanism). The memory of the past is recorded with an *age memory variable* associated with each timed transition. The age memory variable accounts for the work performed by the activity associated with the transition since the time of its last firing. In other words, the age memory variable measures the *cumulative* enabling time of the transition since the last instant of time when it fired.

The three memory policies can be used in timed PN models for different modelling purposes. In the first case (resampling) the work performed by activities associated with transitions that do not fire is lost. This may be adequate for modelling, for example, competing activities of the type one may find in the case of the parallel execution of hypothesis tests. The process that terminates first is the one that verified the test; those hypotheses whose verification was not completed become useless, and the corresponding computations need not be saved. The practical and explicit use of this policy is very limited, but it must be considered because of its theoretical importance in the case of SPNs and GSPNs.

The other two policies are of greater importance from the application viewpoint. They can coexist within the same timed PN model, because of the different semantics that can be assigned to the different transitions of the model. For a detailed discussion on this topic the reader is referred to [2,31].

6.6 Multiple Enabling

Special attention must be paid to the timing semantics in the case of timed transitions with enabling degree larger than one. Different semantics are possible when several tokens are present in the input places of a transition. Borrowing from queueing network terminology, we can consider the following different situations.

1. **Single-server semantics**: a firing delay is set when the transition is first enabled, and new delays are generated upon transition firing if the transition is still enabled in the new marking.
2. **Infinite-server semantics**: every enabling set of tokens is processed as soon as it forms in the input places of the (timed) transition. Its corresponding firing delay is generated at this time, and the timers associated with all these enabling sets run down to zero in parallel.
3. **Multiple-server semantics**: enabling sets of tokens are processed as soon as they form in the input places of the transition up to a maximum degree

of parallelism (say K). For larger values of the enabling degree, the timers associated with new enabling sets of tokens are set only when the number of concurrently running timers decreases below the value of K.

A simple example will help the reader to understand the three semantics. Consider a timed transition T with enabling degree 3. Assume also that the net starts to operate at time 0 with such an enabling degree. The three enablings are associated with three activities whose durations are 3, 2, and 4 time units, respectively. We describe next the detailed behaviour of the net, considering the three different semantics with reference to Fig. 18 that illustrates the firing epochs.

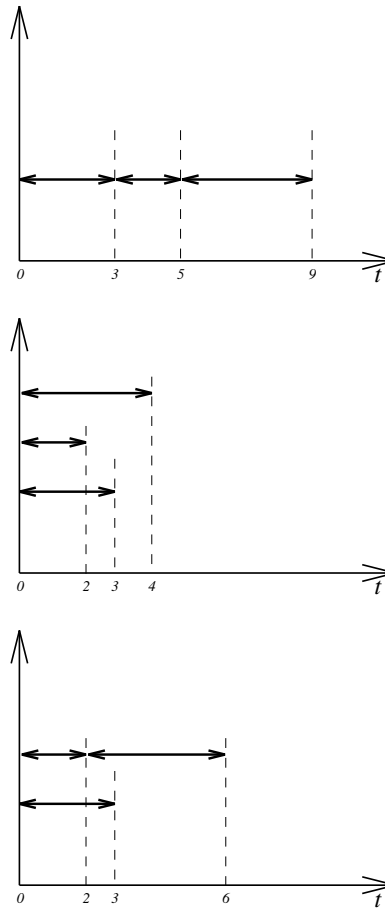


Fig. 18. Firing epochs corresponding to the three different timing semantics

1. Single-server semantics: the serial execution of the activities induces the following sequence of events:
   - t = 0: T_1 is enabled and the first activity starts.
   - t = 3: the first activity (duration 3) ends, T_1 fires and the second activity starts.
   - t = 5: the second activity (duration 2) ends, T_1 fires and the third activity starts.
   - t = 9: the third activity (duration 4) ends and T_1 is disabled.
2. Infinite-server semantics: the parallel execution of the activities induces the following sequence of events:
   - t = 0: T_1 is enabled and all the activities start.
   - t = 2: T_1 fires because of the completion of the second activity (duration 2).
   - t = 3: T_1 fires because of the completion of the first activity (duration 3).
   - t = 4: T_1 fires because of the completion of the third activity (duration 4), and it is disabled.
3. Multiple-server semantics: in this case we assume that the maximum parallelism is $K = 2$. This induces the following sequence of events:
   - t = 0: T_1 is enabled and the first two activities start.
   - t = 2: T_1 fires because of the completion of the second activity (duration 2) thus the third activity can start.
   - t = 3: T_1 fires because of the completion of the first activity (duration 3).
   - t = 6: T_1 fires because of the completion of the third activity (duration 4), and it is disabled.

The introduction of these different firing semantics permits the definition of PN models that are graphically simple without losing any of the characteristics that allow the analysis of their underlying behaviours.

7 Stochastic Petri Nets

Timed Petri nets in which the firing delays are specified by random variables yeld to probabilistic models. The execution of a timed PN model corresponds to a realization of a stochastic point process.

The use of exponential distributions for the definition of temporal specifications is particularly attractive because timed PN in which all the transition delays are exponentially distributed can be mapped onto continuous-time Markov chains (CTMC). In this case the memoryless property of the exponential distribution makes unnecessary the distinction between the distribution of the delay itself, and the distribution of the remaining delay after a change of state.

Stochastic Petri nets are timed (transition) PN with atomic firing and in which transition firing delays are exponentially distributed random variables: each transition t_i is associated with a random firing delay whose probability density function is a negative exponential with rate w_i.

SPNs were originally defined in [37,53] Formally, a SPN model is an 6-tuple

$$\text{SPN} = (P, T, I(.), O(.), W(.), \boldsymbol{m}_0) \tag{12}$$

P, T, $I(.)$, $O(.)$, and $\boldsymbol{m}_0$ have the usual meanings so that the underlying PN model constitutes the structural component of a SPN model.

The function W allows the definition of the stochastic component of a SPN model mapping transitions into real positive functions of the SPN marking. Thus, for any transition t it is necessary to specify a function $W(t, \boldsymbol{m})$. In the case of marking independency, the simpler notation w_k is normally used to indicate $W(t_k)$, for any transition $t_k \in T$. The quantity $W(t_k, \boldsymbol{m})$ (or w_k) is called the "rate" of transition t_k in marking $\boldsymbol{m}$.

In this section we show how SPNs can be converted into Markov chains and how their analysis can be performed to compute interesting performance indices. The construction of the Markov chain associated with a Stochastic Petri Net (SPN) is described first, to set the ground for the subsequent derivation of the probabilistic model associated with a GSPN. Only SPNs with finite state space are considered, as they yield Markov chains that are solvable with standard numerical techniques. More advanced solution methods based on matrix-geometric theory [57] are discussed in [38,57,12].

7.1 The Stochastic Process Associated with a SPN

We have already discussed the motivations behind the work of several authors that led to the proposal of Timed and Stochastic Petri Nets (SPNs). Due to the memoryless property of the exponential distribution of firing delays, it is relatively easy to show [37,53] that SPN systems are isomorphic to continuous time Markov chains (CTMCs). In particular, a k-bounded SPN system can be shown to be isomorphic to a finite CTMC.

Finite State Machine and Marked Graph SPNs - This can be easily seen when the structure of the SPN is that of both a *finite state machine* (no transition has more than one input and one output place) and of a *marked graph* (no place has more than one input and one output transition) with only one token in its initial marking. In this case each place of the net univocally corresponds to a state of the model and the position of the token at a given instant of time identifies the state of the model at that same time. Each place of the net maps into a state of the corresponding CTMC and each transition maps into an arc annotated by the rate of the corresponding firing time distribution. Moreover, if the firing times of the transitions have negative exponential distributions and if the structure of the net is that of a marked graph, the time spent in each place by the net is completely identified by the characteristics of the only transition that may withdraw the token from that place. When the net has the structure of a finite

state machine, conflicts among simultaneously enabled transitions arise since several transitions may share the same input place. Since we are assuming that all the activities have negative-exponentially distributed durations, the CTMC corresponding to the SPN is obtained from the net in a straightforward manner. Again each place of the SPN maps into a state of the corresponding CTMC and each transition of the SPN maps into an arc of the CTMC annotated with the rate of the corresponding firing time distribution. Also in this case the time spent by the net in each place has a negative-exponential distribution, but its rate is given by the sum of the firing rates of all the transitions that withdraw tokens from that place.

More complex situations arise, even in this simple case, when several tokens are allowed in the initial marking. These are due to the fact that places no longer correspond to states of the associated probabilistic models. Moreover, in these cases the behaviours of the probabilistic models are also affected by the service selection policies from the input places as well as by the token selection policies adopted at transition firing moments.

Even the very simple case of two tokens waiting in the only input place of a transition raises a set of interesting questions that must be addressed in developing the corresponding probabilistic model. The first has to do with the speed at which the transition withdraws the tokens from its input place in this situation and thus from the service policies discussed in Section 6.6. In the more general case of assuming a form of *load dependency* in which the firing rate of the transition is a function of its enabling degree, an additional specification must be introduced in the model to define the load dependency function associated with each transition. The second question refers to the selection of the token that is removed from the input place upon the firing of the transition. From a "classical" Petri net point of view, this selection policy is inessential since tokens do not carry any identity. In many applications however, it is convenient to associate a physical meaning with the tokens (e.g., customers), so that questions on their flow through the net can be answered. In these situations, when several tokens are simultaneously present in the input place of a transition, if this is assumed to operate with a single server policy, a question on the queueing policy applied to these tokens becomes interesting. The most natural policy (from a Petri net point of view) is a *random order*. When the firing times are exponentially distributed and when the performance figures of interest are only related to the moments of the number of tokens in the input place of a transition, it is possible to show that many queueing policies yield the same results (e.g., random, FIFO, LCFS). It must however be observed that in other cases the choice of the token selection policy may be important and that policies different from the random one must be explicitly implemented through appropriate net constructions.

SPNs with General Structure - In general, the CTMC associated with a given SPN system is obtained by applying the following simple rules:

1. The CTMC state space $S = \{s_i\}$ corresponds to the reachability set $RS(\boldsymbol{m}_0)$ of the PN associated with the SPN ($\boldsymbol{m}_i \leftrightarrow s_i$).

2. The transition rate from state s_i (corresponding to marking $\boldsymbol{m}_i$) to state s_j ($\boldsymbol{m}_j$) is obtained as the sum of the firing rates of the transitions that are enabled in $\boldsymbol{m}_i$ and whose firings generate marking $\boldsymbol{m}_j$.

Based on these simple rules, it is possible to devise algorithms for the automatic construction of the infinitesimal generator (also called the state transition rate matrix) of the isomorphic CTMC, starting from the SPN description.

Assuming that all the transitions of the net operate with a single-server semantics and marking-independent speeds, and denoting with $\mathbf{Q}$ this matrix, with w_k the firing rate of T_k, and with $E_j(\boldsymbol{m}_i) = \{h : T_h \in E(\boldsymbol{m}_i) \wedge \boldsymbol{m}_i[T_h\rangle \boldsymbol{m}_j\}$ the set of transitions whose firings bring the net from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$, the components of the infinitesimal generator are:

$$q_{ij} = \begin{cases} \sum_{T_k \in E_j(\boldsymbol{m}_i)} w_k & i \neq j \\ -q_i & i = j \end{cases} \tag{13}$$

where

$$q_i = \sum_{T_k \in E(\boldsymbol{m}_i)} w_k \tag{14}$$

Let $\pi(\boldsymbol{m}_i, \tau)$ be the probability that the SPN is in marking $\boldsymbol{m}_i$ at time τ. The Chapman-Kolmogorov equations for the CTMC associated with an SPN are specified by:

$$\frac{d\pi(\boldsymbol{s}_i, \tau)}{d\tau} = \sum_{\boldsymbol{s}_k} \pi(\boldsymbol{s}_k, \tau) q_{kj} \tag{15}$$

In matrix notation this becomes

$$\frac{d\boldsymbol{\pi}(\tau)}{d\tau} = \boldsymbol{\pi}(\tau)\boldsymbol{Q}, \tag{16}$$

whose solution can be formally written as

$$\boldsymbol{\pi}(\tau) = \boldsymbol{\pi}(0) e^{\boldsymbol{Q}\tau} \tag{17}$$

where $\boldsymbol{\pi}(0)$ is the probability of the initial distribution (in our case we usually have $\pi_i(0) = 1$ if $\boldsymbol{m}_i = \boldsymbol{m}_0$ and $\pi_i(0) = 0$ otherwise) and $e^{\boldsymbol{Q}\tau}$ is the matrix exponentiation formally defined by

$$e^{\boldsymbol{Q}\tau} = \sum_{k=0}^{\infty} \frac{(\boldsymbol{Q}\tau)^k}{k!} \tag{18}$$

In this section we consider only SPNs originating homogeneous and ergodic CTMC. A k-bounded SPN system is said to be ergodic if it generates an ergodic CTMC; it is possible to show that a SPN system is ergodic if $\boldsymbol{m}_0$, the initial marking, is a home state (see Section 2).

If the SPN is ergodic, the steady-state probability distribution on its markings exists and is defined as the limit $\boldsymbol{\pi} = \lim_{\tau \to \infty} \boldsymbol{\pi}(\tau)$. Its value can be computed solving the usual system of linear equations:

$$\begin{cases} \boldsymbol{\pi}\ \boldsymbol{Q} &= \mathbf{0} \\ \\ \boldsymbol{\pi}\ \mathbf{1}^{\mathrm{T}} &= 1 \end{cases} \tag{19}$$

where $\mathbf{0}$ is a vector of the same size as $\boldsymbol{\pi}$ and with all its components equal to zero and $\mathbf{1}^{\mathrm{T}}$ is a vector (again of the same size as $\boldsymbol{\pi}$) with all its components equal to one, used to enforce the normalization condition.

To keep the notation simple, in the rest of this section we will use $\pi_i(\tau)$ and π_i instead of $\pi(\boldsymbol{m}_i, \tau)$ and $\pi(\boldsymbol{m}_i)$ to denote the transient and steady state probabilities of marking $\boldsymbol{m}_i$. The *sojourn time* is the time spent by the PN in a given marking $\boldsymbol{m}$. As we already observed, the Markovian property of this model ensures that the sojourn time in the $i-th$ marking is exponentially distributed with rate q_i. The pdf of the sojourn time in a marking corresponds to the pdf of the minimum among the firing times of the transitions enabled in the same marking; it thus follows that the probability that a given transition $T_k \in E(\boldsymbol{m}_i)$ fires (first) in marking $\boldsymbol{m}_i$ can be expressed as follows:

$$P\{T_k | \boldsymbol{m}_i\} \ = \ \frac{w_k}{q_i}. \tag{20}$$

Using the same argument, we can observe that the average sojourn time in marking $\boldsymbol{m}_i$ is given by the following expression:

$$SJ_i \ = \ \frac{1}{q_i}. \tag{21}$$

SPN Performance Indices - The steady-state distribution $\boldsymbol{\pi}$ is the basis for a quantitative evaluation of the behaviour of the SPN that is expressed in terms of performance indices. These results can be computed using a unifying approach in which proper index functions (also called *reward functions*) are defined over the markings of the SPN and an average reward is derived using the steady-state probability distribution of the SPN. Assuming that $r(\boldsymbol{m})$ represents a reward function, the average reward can be computed using the following weighted sum:

$$E[R] \ = \ \sum_{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0)} r(\boldsymbol{m}_i)\ \pi_i \tag{22}$$

Different interpretations of the reward function can be used to compute different performance indices. In particular, the following quantities can be easily computed.

(1) The *probability of a particular condition* of the SPN. Assuming that *condition* $\Upsilon(\boldsymbol{m})$ is *true* only in certain markings of the PN, we can define the following reward function:

$$r(\boldsymbol{m}) \ = \ \begin{cases} 1 & \Upsilon(\boldsymbol{m}) = true \\ \\ 0 & otherwise \end{cases} \tag{23}$$

The desired probability $P\{\Upsilon\}$ is then computed using Equation (22). The same result can also be expressed as:

$$P\{\Upsilon\} = \sum_{\boldsymbol{m}_i \in A} \pi_i \tag{24}$$

where $A = \{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0) : \Upsilon(\boldsymbol{m}_i) = true\}$.

(2) The *expected value of the number of tokens in a given place.* In this case the reward function $r(\boldsymbol{m})$ is simply the value of the marking of that place (say place j):

$$r(\boldsymbol{m}) = n \text{ iff } m(p_j) = n \tag{25}$$

Again this is equivalent to identifying the subset $A(j,n)$ of $RS(\boldsymbol{m}_0)$ for which the number of tokens in place p_j is n ($A(j,n) = \{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0) : \boldsymbol{m}_i(p_j) = n\}$); the expected value of the number of tokens in p_j is given by:

$$E[m(p_j)] = \sum_{n>0} [n\ P\{A(j,n)\}] \tag{26}$$

where the sum is obviously limited to values of $n \leq k$, if the place is k-bounded.

(3) The *mean number of firings per unit of time of a given transition.* Assume that we want to compute the firing frequency of transition T_j (the *throughput* of T_j); observing that a transition may fire only when it is enabled, we have that the reward function assumes the value w_j in every marking that enables T_j:

$$r(\boldsymbol{m}) = \begin{cases} w_j & T_j \in E(\boldsymbol{m}) \\ 0 & otherwise \end{cases} \tag{27}$$

The same quantity can also be computed using the more traditional approach of identifying the subset A_j of $RS(\boldsymbol{m}_0)$ in which a given transition T_j is enabled ($A_j = \{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0) : T_j \in E(\boldsymbol{m}_i)\}$). The mean number of firings of T_j per unit of time is then given by:

$$f_j = \sum_{\boldsymbol{m}_i \in A_j} w_j\ \pi_i \tag{28}$$

These results show that Petri nets can be used not only as a formalism for describing the behaviour of distributed/parallel systems and for assessing their qualitative properties, but also as a tool for computing performance indices that allow the efficiency of these systems to be evaluated.

To illustrate the details of this last analysis step, a simple example is presented in the following section, which encompasses the explicit derivation of the CTMC infinitesimal generator, of the steady-state probability distribution, and of some performance indices.

An Example SPN Model - Consider a simple shared memory multiprocessor system in which two processors must occasionally access a common shared memory. All the processors are assumed to have identical behaviours characterized by a cyclic sequence of local activities, followed by requests and accesses for the common memory. All these actions last for certain amount of times. Additional delays are experienced by a processor that requests to access the common memory while it is busy serving the other processor. Assuming that all timings considered in the example have negative exponential distributions, the SPN model of this system is depicted in Fig. 19. The net comprises seven places and six timed transitions with single-server semantics.

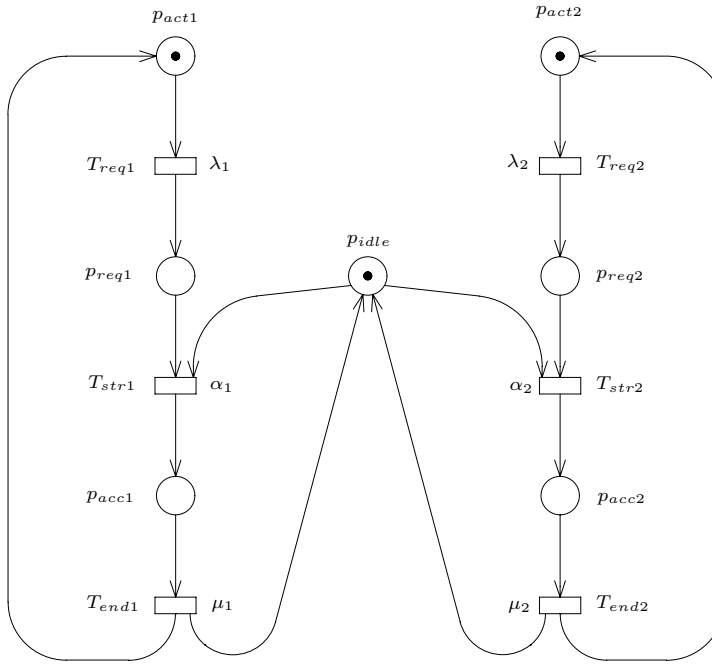


Fig. 19. The SPN description of shared memory system

Starting from the initial marking shown in Fig. 19, in which the two processors are both in a locally active state and the memory is idle ($p_{act1}+p_{act2}+p_{idle}$), a possible evolution of the SPN marking that focuses on the processing cycle of processor 1, may be the following. Processor 1 works locally for an exponentially distributed random amount of time with average $1/\lambda_1$, and then requests an access to the common memory. Transition T_{req1} fires, and the token contained in p_{act1} is removed while a token is added in p_{req1}. Since the common memory is available (place p_{idle} is marked), the acquisition of the memory starts immediately and takes an average of $1/\alpha_1$ units of time to complete; this is represented by the firing of transition T_{str1} whose associated delay has a negative-exponential

distribution with rate α_1; when transition T_{str1} fires, one token is removed from place p_{req1} and another token is deposited into place p_{acc1}, where it stays for the entire time required by the first processor to access the common memory. Such a time lasts on the average $1/\mu_1$ units, and ends when transition T_{end1} fires returning the net to its initial state. Obviously, a similar processing cycle is possible for processor 2 and many interleavings between the activities of the two processors may be described by the evolution of the net.

A conflict exists in the behaviour of this system when both processors want to simultaneously access the common memory, i.e., when transitions T_{str1} and T_{str2} are both enabled. According to Equation (13), in this situation, transition T_{str1} fires with probability:

$$P\{T_{str1}\} = \frac{\alpha_1}{\alpha_1 + \alpha_2} \tag{29}$$

whereas transition T_{str2} fires with probability:

$$P\{T_{str2}\} = \frac{\alpha_2}{\alpha_1 + \alpha_2} \tag{30}$$

Notice that when the two transitions T_{str1} and T_{str2} are both enabled in a given marking M, the speed at which the PN model exits from that marking is the sum of the individual speeds of the two transitions and the conflict is actually resolved only at the moment the first of them fires.

Table 1. Reachability set of SPN of Fig. 19

$\boldsymbol{m}_0$	$= p_{act1} +$				$p_{idle} +$	p_{act2}			
$\boldsymbol{m}_1$	$=$		$p_{req1} +$		$p_{idle} +$	p_{act2}			
$\boldsymbol{m}_2$	$=$			$p_{acc1} +$		p_{act2}			
$\boldsymbol{m}_3$	$=$			$p_{acc1} +$				p_{req2}	
$\boldsymbol{m}_4$	$=$		$p_{req1} +$		$p_{idle} +$			p_{req2}	
$\boldsymbol{m}_5$	$= p_{act1} +$				$p_{idle} +$			p_{req2}	
$\boldsymbol{m}_6$	$= p_{act1} +$								p_{acc2}
$\boldsymbol{m}_7$	$=$		$p_{req1} +$						p_{acc2}

The reachability set of the SPN model of Fig. 19 is listed in Table 1. The reachability graph is shown in Fig. 20, while the corresponding CTMC transition rate diagram is presented in Fig. 21.

8 Generalized Stochastic Petri Nets

Several reasons suggest the introduction of the possibility of using immediate transitions into PN models together with timed transitions. As we observed in

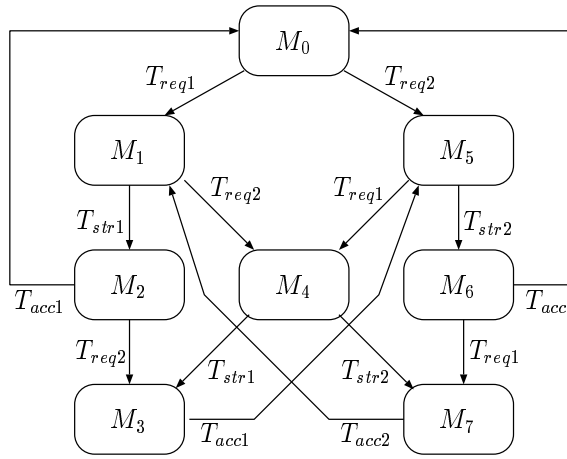


Fig. 20. The reachability graph of the SPN system in Fig. 19

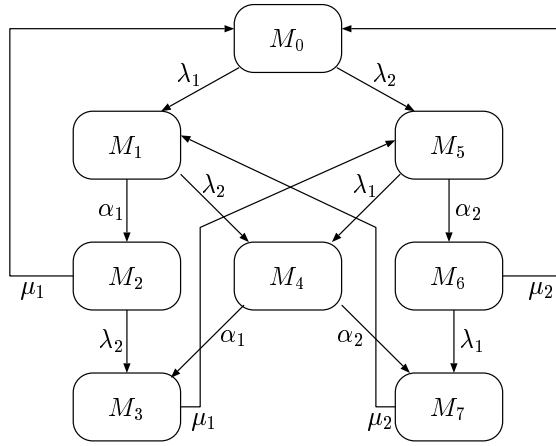


Fig. 21. The state transition rate diagram of the Markov chain associated with the SPN in Fig. 19

Section 6.3 the firing of a transition may describe either the completion of a time-consuming activity, or the verification of a logical condition. It is thus natural to use timed transitions in the former case, and immediate transitions in the latter. Moreover, as we noted in the previous sections, when all transitions are timed the temporal specification of the model must in some cases consider at one time both the timing and the probability inherent in a choice. It seems natural to separate the two aspects in the modelling paradigm, to simplify the model specification. Furthermore, by allowing the use of immediate transitions, some important benefits can be obtained in the model solution. They will be described

in detail later in this section; we only mention here the fact that the use of immediate transitions may significantly reduce the cardinality of the reachability set, and may eliminate the problems due to the presence in the model of timed transitions with rates that differ by orders of magnitude. The latter situation results in so-called "stiff" stochastic processes, that are quite difficult to handle from a numerical viewpoint. On the other hand, the introduction of immediate transitions in an SPN does not raise any significant complexity in the analysis, as we shall see soon.

SPN models in which immediate transitions coexist with timed transitions with race policy and random firing delays with negative exponential pdf are known by the name *generalized SPNs* (GSPNs) [4].

In the graphical representation of GSPNs, immediate transitions are drawn as bars or segments and are denoted by a name that is normally of the form t_x, where x is either a number or a mnemonic string; timed transitions are drawn as (white or black) rectangular boxes, and are denoted by a name that is normally of the form T_x.

Immediate transitions are fired with priority over timed transitions. Thus, if timing is disregarded, the resulting PN model comprises transitions at different priority levels. The adoption of the race policy may seem to implicitly provide the priority of immediate over timed transitions; this is indeed the case in most situations, but the explicit use of priority simplifies the development of the theory. We shall return to this subtle point later in this section.

Recall that markings in the reachability set can be classified as *tangible* or *vanishing*. A marking in which no transition is enabled is tangible. The time spent in any vanishing marking is deterministically equal to zero, while the time spent in tangible markings is positive with probability one.

To describe the GSPN dynamics, we separately observe the timed and the immediate behaviour, hence referring to tangible and vanishing markings, respectively. Let us start with the timed dynamics (hence with tangible markings); this is identical to the dynamics in SPNs, that was described before. We can assume that each timed transition possesses a timer. The timer is set to a value that is sampled from the negative exponential pdf associated with the transition, when the transition becomes enabled for the first time after firing. During all time intervals in which the transition is enabled, the timer is decremented. Transitions fire when their timer reading goes down to zero.

With this interpretation, each timed transition can be used to model the execution of some activity in a distributed environment; all enabled activities execute in parallel (unless otherwise specified by the PN structure) until they complete. At completion time, activities induce a change of the system state, only as regards their local environment. No special mechanism is necessary for the resolution of timed conflicts: the temporal information provides a metric that allows the conflict resolution.

In the case of vanishing markings, the GSPN dynamics consumes no time: everything takes place instantaneously. This means that if only one immediate transition is enabled, it fires, and the following marking is produced. If sev-

eral immediate transitions are enabled, a metric is necessary to identify which transition will produce the marking modification. Actually, the selection of the transition to be fired is relevant only in those cases in which a conflict must be resolved: if the enabled transitions are concurrent, they can be fired in any order. For this reason, GSPNs associate *weights* with immediate transitions belonging to the same conflict set.

For the time being, let us consider only free-choice conflict sets; the case of non-free-choice conflict sets will be considered later on, but we can anticipate at this point that it can be tackled in a similar manner by exploiting the definition of ECS introduced in Section 5.1. The transition weights are used to compute the firing probabilities of the simultaneously enabled transitions comprised within the conflict set. The restriction to free-choice conflict sets guarantees that transitions belonging to different conflict sets cannot disable each other, so that the selection among transitions belonging to different conflict sets is not necessary.

We can thus observe a difference between the specification of the temporal information for timed transitions and the specification of weights for immediate transitions. The temporal information associated with a timed transition depends only on the characteristics of the activity modelled by the transition. Thus, the temporal specification of a timed transition requires no information on the other (possibly conflicting) timed transitions, or on their temporal characteristics. On the contrary, for immediate transitions, the specification of weights must be performed considering at one time all transitions belonging to the same conflict set. Indeed, weights are normalized to produce probabilities by considering all enabled transitions within a conflict set, so that the specification of a weight, independent of those of the other transitions in the same conflict set, is not possible.

8.1 Some Extensions

Some additional features can be included in a GSPN model, with an advantage in the power of the modelling paradigm and little increase in the analysis complexity. These extensions are the possibility of using non-free-choice conflicts of immediate transitions, the availability of multiple priority levels for immediate transitions, and the marking-dependency of transition annotations (rates for timed transitions and weights for immediate transitions).

The availability of multiple priority levels for immediate transitions, and the marking-dependency of transition annotations has quite a beneficial impact on the modelling power, and hardly any impact on the complexity of the model specification and analysis. In particular, the availability of multiple priority levels for immediate transitions permits the simple specification of complex sequential selection algorithms, while the marking-dependency of transition annotations allows the development of compact models in which the behaviours of a number of different entities are synthetically described.

Employing non-free-choice conflicts of immediate transitions, the user has the possibility of describing a much wider range of dynamic behaviours in vanishing markings, but he must be able to correctly associate the immediate transitions

with the metrics that define the probabilistic conflict resolution. This requires the knowledge of the sets of simultaneously enabled non-concurrent immediate transitions in any vanishing marking. This knowledge may not be easy to obtain without the generation of the reachability set, which is however very costly in most cases. The definition of extended conflict sets (ECSs) was introduced in Section 5 to provide the user with the information on the sets of transitions that *may* be in effective conflict (either direct or indirect) in a marking, thus helping in the definition of weights.

One extension that is not possible within the GSPN framework is the introduction of more general forms of pdf for the firing delays associated with transitions. Nevertheless, also in this respect, the availability of immediate and exponential transitions in one modelling paradigm can be exploited for the construction of somewhat more general pdfs in the description of the duration of the real system activities (see [6] for a detailed discussion of this topic).

8.2 The Definition of a GSPN Model

GSPNs were originally defined in [4]. The definition was later improved to better exploit the structural properties of the modelling paradigm [3]. The definition we present here is based on the version contained in this second proposal.

Formally, a GSPN model is an 8-tuple

$$\text{GSPN} = (P, T, \Pi(.), I(.), O(.), H(.), W(.), \boldsymbol{m}_0) \tag{31}$$

where $\text{PN}_\pi = (P, T, \Pi(.), I(.), O(.), H(.), \boldsymbol{m}_0)$ is the marked PN with priority underlying the GSPN and $W(.)$ is a function defined on the set of transitions

Timed transitions are associated with priority zero, whereas all other priority levels are reserved for immediate transitions.

The underlying PN model constitutes the structural component of a GSPN model, and it must be confusion-free (see Section 5.1) at priority levels greater than zero (i.e., in subnets of immediate transitions).

The function W allows the definition of the stochastic component of a GSPN model. In particular, it maps transitions into real positive functions of the GSPN marking. Thus, for any transition t it is necessary to specify a function $W(t, \boldsymbol{m})$. In the case of marking independency, the simpler notation w_k is normally used to indicate $W(t_k)$, for any transition $t_k \in T$. The quantity $W(t_k, \boldsymbol{m})$ (or w_k) is called the "rate" of transition t_k in marking $\boldsymbol{m}$ if t_k is timed, and the "weight" of transition t_k in marking M if t_k is immediate.

Since in any marking all firing delays of timed transitions have a negative exponential pdf, and all the delays are independent random variables, the sojourn time in a tangible marking is a random variable with a negative exponential pdf whose rate is the sum of the rates of all enabled timed transitions in that marking. In the case of vanishing markings, the weights of the immediate transitions enabled in an ECS can be used to determine which immediate transition will actually fire, if the vanishing marking enables more than one conflicting immediate transition.

When transitions belonging to several different ECSs are simultaneously enabled in a vanishing marking, as we already explained, the choice among these transitions is irrelevant. It must be emphasized that the irrelevance in the order of transition firings is an important consequence of the restriction that subnets of immediate transitions must be confusion-free. The restriction to confusion-free immediate subnets also has a beneficial impact on the model definition, since the association of weights with immediate transitions requires only the information about ECSs, not about reachable markings. For each ECS the analyst thus defines a *local* association of weights from which probabilities are then derived.

8.3 Some Fine Points

Let us return to the points we raised in the previous sections, but left unanswered. These are:

1. the need for an explicit priority of immediate over timed transitions;
2. the irrelevance of the distinction between resampling, enabling memory, and age memory;
3. the impossibility for two timers to expire at the same time.

All three points relate to the properties of the negative exponential pdf. This distribution characterizes a continuous random variable, hence it is a continuous function defined in the interval $[0, \infty)$, that integrates to one. The lack of discontinuities in the function makes the probability of any specific value x being sampled equal to zero (however, obviously, the probability that a value is sampled between two distinct values $x_1 \geq 0$ and $x_2 > x_1$ is positive).

Let us look now at the three previous points in order.

1. Consider a free-choice conflict set comprising an immediate and a timed transition, and assume for a moment that priority does not exist. The race policy makes the immediate transition always win, except for the case in which a zero delay is sampled from the negative exponential pdf. Although the probability of selecting the value zero is null, some problem may arise when the conflict set is enabled infinitely often in a finite time interval. For example, in the case of Fig. 22, the timed and the immediate transitions are always enabled, because the firing of the immediate transition t_2 does not alter the PN marking. The situation is changed only when the timed transition T_1 fires. This happens with probability one in time zero, possibly after an infinite number of firings of the immediate transition. To avoid these (sometimes strange) limiting behaviours, the priority of immediate over timed transitions was introduced in the GSPN definition. This makes the timed transition T_1 in Fig. 22 never enabled.
2. The *memoryless property* of the negative exponential pdf, ensures that at any time instant, the residual time until a timer associated with a transition expires is statistically equivalent to the originally sampled timer reading. Thus, whether a new timer value is set at every change of marking, or at every instant a transition becomes enabled after disabling, or after firing,

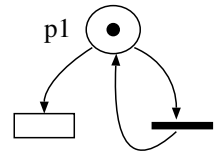


Fig. 22. A conflict set comprising a timed and an immediate transition

makes no difference from the point of view of the probabilistic metrics of the GSPN.

3. Since the probability that a sample extracted from a negative exponential pdf takes a specific value x equals zero, the probability of two timers expiring at the same time is null. Indeed, given the value sampled by the first timer, the probability that the second one samples the same value is zero.

8.4 The Stochastic Process Associated with a GSPN

As we have just observed, GSPNs adopt the same firing policy of SPNs; when several transitions are enabled in the same marking, the probabilistic choice of the transition to fire next depends on parameters that are associated with these same transitions and that are not functions of time. The general expression for the probability that a given (timed or immediate) transition t_k, enabled in marking $\boldsymbol{m}_i$, fires is:

$$P\{t_k|\boldsymbol{m}_i\} = \frac{w_k}{q_i} \tag{32}$$

where q_i is the quantity defined by Equation (14). Equation (32) represents the probability that transition t_k fires first, and is identical to Equation (20) for SPNs, with a difference in the meaning of the parameters w_k. When the marking is vanishing, the parameters w_k are the weights of the immediate transitions enabled in that marking and define the selection policy used to make the choice. When the marking is tangible, the parameters w_k of the timed transitions enabled in that marking are the rates of their associated negative exponential distributions. The average sojourn time in vanishing markings is zero, while the average sojourn time in tangible markings is given by Equation (21).

Observing the evolution of the GSPN system, we can notice that the distribution of the sojourn time in an arbitrary marking can be expressed as a composition of negative exponential and deterministically zero distributions: we can thus recognize that the marking process $\{\mathcal{M}(\tau), \tau \geq 0\}$ is a semi-Markov stochastic process.

When several immediate transitions are enabled in the same vanishing marking, deciding which transition to fire first makes sense only in the case of conflicts. If these immediate transitions do not "interfere" they could be fired simultaneously and the choice of firing only one of them at a time becomes an operational rule of the model that hardly relates with the actual characteristics of the DEDS

we are modelling. In this case the selection is inessential from the point of view of the overall behaviour of the net.

Assuming that the GSPN is not confused, the computation of the ECSs of the net corresponds to partitioning the set of immediate transitions into equivalence classes such that transitions of the same partition may be in conflict among each other in possible markings of the net, while transitions of different ECSs behave in a truly concurrent manner.

When transitions belonging to the same ECS are the only ones enabled in a given marking, one of them (say transition t_k) is selected to fire with probability:

$$P\{t_k|\boldsymbol{m}_i\} = \frac{w_k}{\omega_k(\boldsymbol{m}_i)} \tag{33}$$

where $\omega_k(\boldsymbol{m}_i)$ is the weight of ECS(t_k) in marking $\boldsymbol{m}_i$ and is defined as follows:

$$\omega_k(\boldsymbol{m}_i) = \sum_{t_j \in [\mathrm{ECS}(t_k) \wedge E(\boldsymbol{m}_i)]} w_j \tag{34}$$

Within the ECS we may have transitions that are in direct as well as in indirect conflicts. This means that the firing selection probabilities may be different for the same transition in different markings. Equation (33) however ensures that if we have two transitions (say transitions t_i and t_j), both enabled in two different markings (say markings $\boldsymbol{m}_r$ and $\boldsymbol{m}_s$), the ratios between the firing probabilities of these two transitions in these two markings remain constant and in particular equal to the ratio between the corresponding weights assigned at the moment of the specification of the model.

During the evolution of a GSPN, it may happen that several ECSs are simultaneously enabled in a vanishing marking. According to the usual firing mechanism of Petri nets, we should select the transition to fire by first non-deterministically choosing one of the ECSs and then a transition within it. The assumption that the GSPN is not confused guarantees that the way in which the choice of the ECS is performed is irrelevant with respect to the associated stochastic process. One possibility is that of computing the weight of the ECS by adding the parameters of all the enabled transitions that belong to that ECS and of using this weight to select the ECS with a method suggested by Equation (32). A simple derivation shows that the use of this method implies that the selection of the immediate transition to be fired in a vanishing marking can be performed with the general formula (32) that was originally derived for timed transitions (and thus for tangible markings) only [3,21]. Moreover, it is possible to show that if we consider the probabilities associated with the many different sequences of immediate transitions whose firings lead from a given vanishing marking to a target tangible one, they turn out to be all equal [3].

This last property strongly depends on the absence of confusion in the GSPN; the fact that the presence of confused subnets of immediate transitions within a GSPN is an undesirable feature of the model can also be explained considering the following example.

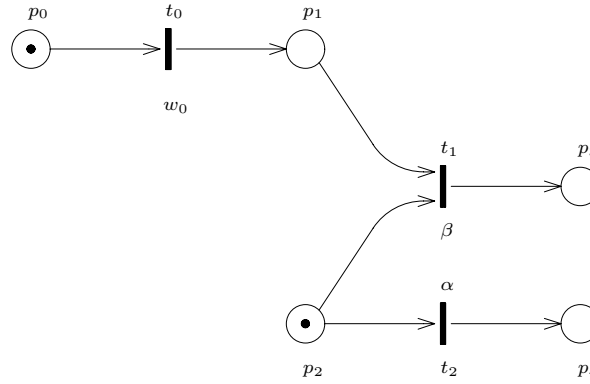


Fig. 23. Example of a simple confused GSPN model

Suppose that a subnet is identified as confused using the structural confusion condition of Section 5.1. Suppose also that the subnet has the structure depicted in Fig. 23. Given the initial marking $\boldsymbol{m}_0 = (p_0 + p_2)$, markings $\boldsymbol{m}_1 = p_3$ and $\boldsymbol{m}_2 = (p_1 + p_4)$ are reached with total probabilities

$$P\{\boldsymbol{m}_0, \boldsymbol{m}_1\} = \frac{w_0}{(w_0 + \alpha)} \frac{\beta}{(\alpha + \beta)} \qquad P\{\boldsymbol{m}_0, \boldsymbol{m}_2\} = \frac{\alpha}{(w_0 + \alpha)} \quad .$$

From these expressions we can see that, although the picture suggests transitions t_0 and t_2 as concurrent, and transitions t_1 and t_2 as members of a conflict set, the first choice of firing either t_0 or t_2 is actually crucial for the possibility of reaching the desired markings. In this case the value assigned by the analyst to w_0 becomes fundamental for the quantitative evaluation of the model.

Much more intriguing however is the behaviour of the subnet in Fig. 24(b) that differs from that of Fig. 24(a) for the simple addition of place p_a and transition t_a between transition t_0 and place p_1. Given the initial marking $\boldsymbol{m}_0 = (p_0 + p_2)$, lmarkings $\boldsymbol{m}_1 = p_3$ and $\boldsymbol{m}_2 = (p_1 + p_4)$ are reached in the case of the subnet of Fig. 24(b) with total probabilities

$$P\{\boldsymbol{m}_0, \boldsymbol{m}_1\} = \frac{w_0}{(w_0 + \alpha)} \frac{w_a}{(w_a + \alpha)} \frac{\beta}{(\alpha + \beta)}$$

and

$$P\{\boldsymbol{m}_0, \boldsymbol{m}_2\} = \frac{\alpha}{(w_0 + \alpha)} + \frac{w_0}{(w_0 + \alpha)} \frac{\alpha}{(w_a + \alpha)} \quad .$$

From a modelling point of view, there are good reasons to consider the two subnets of Figs. 24(a) and 24(b) equivalent since the sequence of immediate actions represented by t_0 and t_a of Fig. 24(b) should be reducible to transition t_0 of Fig. 24(a) without affecting the behaviour of the model. Instead, the difference among the total probabilities that we have just computed shows that, in the case of confused models, the trivial action of splitting an atomic (and instantaneous) action in two has drastic effects not only on the graphical description of the

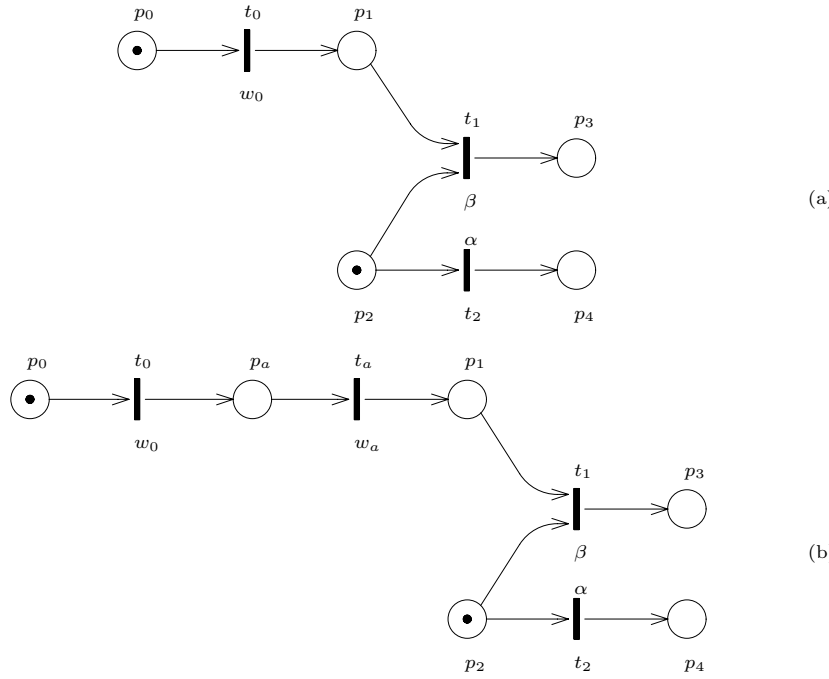


Fig. 24. Comparison of two simple confused GSPN systems

model, but also (and more important) on the values of the results obtained from its quantitative evaluation.

Marking Dependency - The firing times and the weights that we have considered so far were assumed to be independent of the marking of the GSPN. In principle, however, it is possible to work with transition parameters that are marking-dependent as pointed out at the beginning of this section. When one or more timed transitions are enabled in a given marking, we can compute the distribution of the sojourn time in that marking, as well as the probability of the transition that fires first, using negative exponential distributions whose firing rates may depend on that specific marking. Similarly, the selection of the immediate transition that fires in a vanishing marking enabling several immediate transitions can be computed using weighting factors that may be marking-dependent. In all these cases, Equations (21), (32), (33), and (34) can be generalized assuming that all the parameters are functions of the marking for which they are computed.

In practice, the generality allowed by this extension (which was assumed by the original GSPN proposal [4]) is in contrast with the claim of GSPNs as a high-level language for the description of complex systems. In fact, while the construction of a GSPN model requires a local view of the behaviour of the real system, the specification of (general) marking-dependent parameters requires

the analyst to be aware of the possible (global) states of the system. Moreover, a dependency of the weights of conflicting immediate transitions on the marking of places that are not part of the input set of any of the transitions comprised in their ECS could lead to a new form of "confusion" that is not captured by the definitions contained in Section 5.1 and discussed in the previous section.

For this reason, a restriction of the type of marking-dependency allowed in GSPN models was informally proposed in [21]. A definition of marking dependency that satisfies these restrictions can be obtained by allowing the specification of marking dependent parameters as the product of a nominal rate (or weight in the case of immediate transitions) and of a dependency function defined in terms of the marking of the places that are connected to a transition through its input and inhibition functions.

Denote with $\boldsymbol{m}_{/t}$ the restriction of a generic marking M to the input and inhibition sets of transition t:

$$\boldsymbol{m}_{/t} \;=\; \bigcup\nolimits_{p\in({}^{\bullet}t\cup t^{\circ})} m(p) \tag{35}$$

Let $f(\boldsymbol{m}_{/t})$ be the marking dependency function that assumes positive values every time transition t is enabled in M; using this notation, the marking dependent parameters may be defined in the following manner:

$$\begin{cases} \mu_i(\boldsymbol{m}) \;=\; f(\boldsymbol{m}_{/T_i})\; w_i & \text{in case of firing rates,} \\ \\ \omega_j(\boldsymbol{m}) \;=\; f(\boldsymbol{m}_{/t_j})\; w_j & \text{in case of weights.} \end{cases} \tag{36}$$

Multiple-servers and infinite-servers can be represented as special cases of timed transitions with marking dependent firing rates that are consistent with this restriction. In particular, a timed transition T_i with a negative-exponential delay distribution with parameter w_i and with an infinite-server policy, has a marking dependency function of the following form:

$$f(\boldsymbol{m}_{/T_i}) \;=\; \boldsymbol{e}_i(\boldsymbol{m}) \tag{37}$$

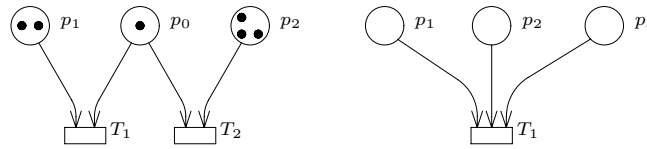
where $\boldsymbol{e}_i(\boldsymbol{m})$ is the enabling degree of transition T_i in marking $\boldsymbol{m}$. Similarly, a timed transition T_i with multiple-server policy of degree K has a marking dependency function defined in the following way:

$$f(\boldsymbol{m}_{/T_i}) \;=\; min\;(\boldsymbol{e}_i(\boldsymbol{m}),\; K) \tag{38}$$

Other interesting situations can be represented using the same technique, Fig. 25 depicts two such cases. In Fig. 25(a), we have a situation of two competing infinite servers: transition T_1 fires with rate $w_1 m(p_1)$ if place p_0 is marked; similarly for transition T_2. Obviously both transitions are interrupted when the first of the two fires removing the token from place p_0.

Using $\delta(x)$ to represent the following step function:

$$\delta(x) \;=\; \begin{cases} 0 & x \;=\; 0 \\ \\ 1 & x \;>\; 0 \end{cases} \tag{39}$$



(a) (b)

Fig. 25. Examples of complex marking dependency situations

we can define the marking dependency function as follows:

$$\begin{cases} f(\boldsymbol{m}_{/T_1}) = \delta(ED(T_1, \boldsymbol{m}))m(p_1) \\ \qquad\qquad \text{and similarly} \\ f(\boldsymbol{m}_{/T_2}) = \delta(ED(T_2, \boldsymbol{m}))m(p_2) \end{cases} \tag{40}$$

Fig. 25(b) represents instead a server whose speed depends on a linear combination of the markings of all its input places. In this case the marking dependency function may assume the form:

$$f(\boldsymbol{m}_{/T_1}) = \delta(\boldsymbol{e}_1(\boldsymbol{m})) \sum_{p \in {}^\bullet T_1} \alpha_p m(p) \qquad \alpha_p \geq 0, \quad \sum_{p \in {}^\bullet T_1} \alpha_p = 1 \tag{41}$$

8.5 Numerical Solution of GSPN Systems

The stochastic process associated with a k-bounded GSPN system with $\boldsymbol{m}_0$ as its home state can be classified as a finite state space, stationary (homogeneous), irreducible, and continuous-time semi-Markov process.

Semi-Markov processes can be analysed identifying an embedded (discrete-time) Markov chain that describes the transitions from state to state of the process. In the case of GSPNs, the embedded Markov chain (EMC) can be recognized disregarding the concept of time and focusing the attention on the set of states of the semi-Markov process. The specifications of a GSPN are sufficient for the computation of the transition probabilities of such a chain.

Let RS, TS, and VS indicate the state space (the reachability set), the set of tangible states (or markings) and the set of vanishing markings of the stochastic process, respectively. The following relations hold among these sets:

$$RS = TS \bigcup VS, \qquad TS \bigcap VS = \emptyset.$$

The transition probability matrix $\boldsymbol{U}$ of the EMC can be obtained from the specification of the model using the following expression:

$$u_{ij} = \frac{\sum_{T_k \in E_j(\boldsymbol{m}_i)} w_k}{q_i} \tag{42}$$

n this way, except for the diagonal elements of matrix $\boldsymbol{U}$, ll the other transition probabilities of the EMC an be computed using quation (32) independently of whether the transition to be onsidered is timed or immediate, according to the discussion contained t the end of Section 8.4.

By ordering the markings so that the vanishing ones correspond to the first entries of the matrix and the tangible ones to the last, the transition probability matrix $\boldsymbol{U}$ can be decomposed in the following manner:

$$\boldsymbol{U} \;=\; \boldsymbol{A} + \boldsymbol{B} \;=\; \begin{bmatrix} \boldsymbol{C} & \boldsymbol{D} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \boldsymbol{E} & \boldsymbol{F} \end{bmatrix} \tag{43}$$

The elements of matrix $\boldsymbol{A}$ correspond to changes of markings induced by the firing of immediate transitions; in particular, those of submatrix $\boldsymbol{C}$ are the probabilities of moving from vanishing to vanishing markings, while those of $\boldsymbol{D}$ correspond to transitions from vanishing to tangible markings. Similarly, the elements of matrix $\boldsymbol{B}$ correspond to changes of markings caused by the firing of timed transitions: $\boldsymbol{E}$ accounts for the probabilities of moving from tangible to vanishing markings, while $\boldsymbol{F}$ comprises the probabilities of remaining within tangible markings.

Indicating with $\boldsymbol{\psi}(n)$ the probability distribution of the EMC at step n (i.e., after n (state-) transitions performed by the EMC), we can compute this quantity using the following expression

$$\boldsymbol{\psi}(n) \;=\; \boldsymbol{\psi}(0)\boldsymbol{U}^n \tag{44}$$

where, as usual, $\boldsymbol{\psi}(0)$ represents the initial distribution of the EMC. The steady-state probability distribution $\boldsymbol{\psi}$ can be obtained as the solution of the system of linear equations

$$\begin{cases} \boldsymbol{\psi} \;=\; \boldsymbol{\psi}\,\boldsymbol{U} \\ \boldsymbol{\psi}\,\mathbf{1}^{\mathrm{T}} \;=\; 1 \end{cases} \tag{45}$$

The steady-state probability distribution of the EMC, can be interpreted in terms of numbers of (state-) transitions performed by the EMC. In fact, $1/\psi_i$ is the mean recurrence time for state s_i (marking $\boldsymbol{m}_i$) measured in number of transition firings. The steady-state probability distribution of the stochastic process associated with the GSPN system is thus obtained by weighting each entry ψ_i with the sojourn time of its corresponding marking SJ_i and by normalizing the whole distribution.

The solution method outlined so far, is computationally acceptable whenever the size of the set of vanishing markings is small (compared with the size of the set of tangible markings). However, this method requires the computation of the steady-state probability of each vanishing marking that is known a priori to be null. Moreover, vanishing markings, by enlarging the size of the transition

probability matrix $\boldsymbol{U}$, tend to make the solution more expensive and in some cases even impossible to obtain.

In order to restrict the solution to quantities directly related with the computation of the transient and steady-state probabilities of tangible markings, we must reduce the model by computing the total transition probabilities among tangible markings only, thus identifying a Reduced EMC (REMC).

To illustrate the method of reducing the EMC by removing the vanishing markings, consider first the example of Fig. 26. This system contains two free-choice conflicts corresponding to transitions T_1 , T_2, and t_1, t_2, respectively. From the initial marking $\boldsymbol{m}_i = p_1$, the system can move to marking $\boldsymbol{m}_j = p_3$ following two different paths. The first corresponds to the firing of transition T_1, that happens with probability $\frac{\mu_1}{(\mu_1+\mu_2)}$, and that leads to the desired (target) marking $\boldsymbol{m}_j$ in one step only. The second corresponds to selecting transition T_2 to fire first, followed by transition t_1. The first of these two events happens with probability $\frac{\mu_2}{(\mu_1+\mu_2)}$, and the second with probability $\frac{\alpha}{(\alpha+\beta)}$. The total probability of this second path from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ amounts to $\frac{\mu_2}{(\mu_1+\mu_2)}\frac{\alpha}{(\alpha+\beta)}$. Notice that firing transition T_2 followed by transition t_2 would lead to a different marking (in this case the initial one). Firing transition T_2 leads the system into an intermediate (vanishing) marking $\boldsymbol{m}_r$. The total probability of moving from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$ is thus in this case:

$$u'_{ij} = \frac{\mu_1}{(\mu_1+\mu_2)} + \frac{\mu_2}{(\mu_1+\mu_2)}\frac{\alpha}{(\alpha+\beta)} \tag{46}$$

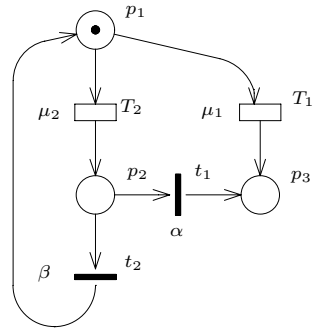


Fig. 26. A GSPN system with multiple paths between tangible markings

In general, upon the exit from a tangible marking, the system may "walk" through several vanishing markings before ending in a tangible one. To make the example more interesting, and to illustrate more complex cases, let us modify the net of Fig. 26 as depicted in Fig. 27. In this new case the system can move from marking $\boldsymbol{m}_1 = p_1$ to marking $\boldsymbol{m}_2 = p_2$ following four different paths. The first corresponds again to firing transition T_1 and thus to a direct move from the

initial marking to the target one. This happens with probability $\frac{\mu_1}{(\mu_1+\mu_2+\mu_3)}$. The second path corresponds to firing transition T_2 followed by t_1 (total probability $\frac{\mu_2}{(\mu_1+\mu_2+\mu_3)}\ \frac{\alpha}{(\alpha+\beta)}$); the third path corresponds to firing transition T_3 followed by transition t_4 (total probability $\frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\ \frac{\gamma}{(\gamma+\delta)}$). Finally, the last path corresponds to firing transition T_3 followed by transition t_3 and then by transition t_1 which happens with probability $\frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\ \frac{\delta}{(\gamma+\delta)}\ \frac{\alpha}{(\alpha+\beta)}$.

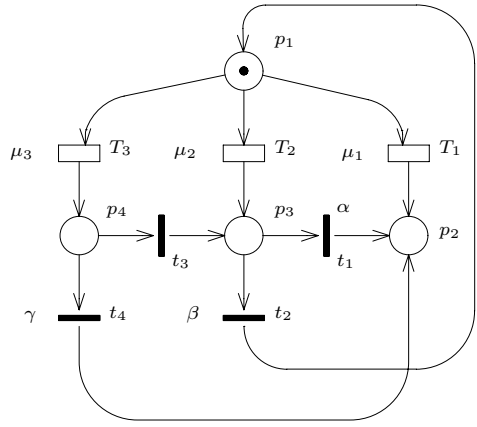


Fig. 27. A GSPN system with several multiple paths between tangible markings

In this case the total probability of moving from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$ becomes:

$$u'_{ij} = \frac{\mu_1}{(\mu_1+\mu_2+\mu_3)} + \frac{\mu_2}{(\mu_1+\mu_2+\mu_3)}\ \frac{\alpha}{(\alpha+\beta)} + \frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\ \frac{\gamma}{(\gamma+\delta)} + \frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\ \frac{\alpha}{(\alpha+\beta)}\ \frac{\delta}{(\gamma+\delta)} \quad (47)$$

In general, recalling the structure of the $\boldsymbol{U}$ matrix, a direct move from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$ corresponds to a non-zero entry in block $\boldsymbol{F}$ ($f_{ij} \neq 0$), while a path from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ via two intermediate vanishing markings corresponds to the existence of

1. a non-zero entry in block $\boldsymbol{E}$ corresponding to a move from $\boldsymbol{m}_i$ to a generic intermediate marking $\boldsymbol{m}_r$;
2. a non-zero entry in block $\boldsymbol{C}$ from this generic state $\boldsymbol{m}_r$ to another arbitrary vanishing marking $\boldsymbol{m}_s$;
3. a corresponding non-zero entry in block $\boldsymbol{D}$ from $\boldsymbol{m}_s$ to $\boldsymbol{m}_j$.

These informal considerations are precisely captured by the following formula:

$$u'_{ij} = f_{ij} + \sum_{r:\boldsymbol{m}_r \in VS} e_{ir} P\{r \to s\}\ d_{sj} \quad (48)$$

where $P\{r \to s\}\, d_{sj}$ is the probability that the net moves from vanishing marking $\boldsymbol{m}_r$ to tangible marking $\boldsymbol{m}_j$ in an arbitrary number of steps, following a path through vanishing markings only.

In order to provide a general and efficient method for the computation of the state transition probability matrix $\boldsymbol{U}'$ of the REMC, we can observe that Equation (48) can be rewritten in matrix notation in the following form:

$$\boldsymbol{U}' \;=\; \boldsymbol{F} \;+\; \boldsymbol{E}\,\boldsymbol{G}\,\boldsymbol{D} \tag{49}$$

where each entry g_{rs} of matrix $\boldsymbol{G}$ represents the probability of moving from vanishing marking $\boldsymbol{m}_r$ to vanishing marking $\boldsymbol{m}_s$ in any number of steps, but without hitting any intermediate tangible marking. $\boldsymbol{G}$ can be expressed with the following formula:

$$\boldsymbol{G} \;=\; \sum_{n=0}^{\infty} \boldsymbol{C}^n$$

In the computation of $\boldsymbol{C}^n$, two possibilities may arise. The first corresponds to the situation in which there are no loops among vanishing markings. This means that for any vanishing marking $\boldsymbol{m}_r \in VS$ there is a value n_{0r} such that any sequence of transition firings of length $n \geq n_{0r}$ starting from such marking must reach a tangible marking $\boldsymbol{m}_j \in TS$. In this case

$$\exists n_0 \;:\quad \forall\;\; n \geq n_0 \qquad \boldsymbol{C}^n \;=\; 0$$

and

$$\boldsymbol{G} \;=\; \sum_{k=0}^{\infty} \boldsymbol{C}^k \;=\; \sum_{k=0}^{n_0} \boldsymbol{C}^k$$

The second corresponds to the situation in which there are possibilities of loops among vanishing markings, so that the GSPN may remain "trapped" within a set of vanishing markings. In this case the irreducibility property of the semi-Markov process associated with the GSPN system ensures that the following results hold [69]:

$$\lim_{n\to\infty} \boldsymbol{C}^n \;=\; 0$$

so that

$$\boldsymbol{G} \;=\; \sum_{k=0}^{\infty} \boldsymbol{C}^k \;=\; [\boldsymbol{I} \;-\; \boldsymbol{C}]^{-1}.$$

We can thus write (see [5,4] for details):

$$\boldsymbol{H} \;=\; \begin{cases} \left(\sum_{k=0}^{n_0} \boldsymbol{C}^k \right) \boldsymbol{D} & \textit{no loops among vanishing states} \\ [\boldsymbol{I} \;-\; \boldsymbol{C}]^{-1}\, \boldsymbol{D} & \textit{loops among vanishing states} \end{cases}$$

from which we can conclude that an explicit expression for the desired total transition probability among any two tangible markings is:

$$u'_{ij} \;=\; f_{ij} \;+\; \sum_{r \in VS} e_{ir}\; h_{rj} \qquad \forall \quad i,j \in TS$$

The transition probability matrix of the REMC can thus be expressed as

$$\boldsymbol{U}' \;=\; \boldsymbol{F} \;+\; \boldsymbol{E}\;\boldsymbol{H} \tag{50}$$

Denoting again with $\boldsymbol{\psi}'(n)$ the probability distribution of the REMC at step n (i.e., after the firing of n timed transitions), we can compute this quantity using the following expression

$$\boldsymbol{\psi}'(n) \;=\; \boldsymbol{\psi}'(0)\boldsymbol{U}'^{n} \tag{51}$$

The steady-state probability distribution $\boldsymbol{\psi}'$ can be obtained as the solution of the system of linear equations

$$\begin{cases} \boldsymbol{\psi}' \;=\; \boldsymbol{\psi}'\;\boldsymbol{U}' \\ \\ \boldsymbol{\psi}'\;\mathbf{1}^{\mathrm{T}} \;=\; 1. \end{cases} \tag{52}$$

The stationary probability distribution associated with the set of tangible markings is thus readily obtained by means of their average sojourn times (see Equation (21)) using a procedure similar vto that outlined before for the EMC method.

The construction of the REMC defined over the set of tangible markings TS implies that a transformation exists of the semi-Markov process associated with every GSPN system into a CTMC. The steady-state probability distribution over the tangible markings can thus be also obtained by a direct solution of this CTMC. In our case, the infinitesimal generator $\boldsymbol{Q}'$ of the CTMC associated with a GSPN can be constructed from the transition probability rate matrix $\boldsymbol{U}'$ of the REMC by dividing each of its rows by the mean sojourn time of the corresponding tangible marking. To conform with the standard definition of the infinitesimal generators, the diagonal elements of $\boldsymbol{Q}'$ are set equal to the negative sum of the off-diagonal components:

$$q'_{ij} \;=\; \begin{cases} \frac{1}{SJ_i}\; u'_{ij} & i \neq j \\ \\ -\sum_{j \neq i} q'_{ij} & i = j \end{cases} \tag{53}$$

This result shows that GSPNs, like SPNs, can be analysed by solving properly associated CTMCs. This obviously implies that the transient state probability distribution is the solution of the following differential equation

$$\frac{d\boldsymbol{\pi}'(\tau)}{d\tau} \;=\; \boldsymbol{\pi}'(\tau)\boldsymbol{Q}' \tag{54}$$

while the steady-state probability distribution over the tangible markings requires the solution of the following system of linear equations:

$$\begin{cases} \boldsymbol{\pi}'\;\boldsymbol{Q}' \;=\; \mathbf{0} \\ \\ \boldsymbol{\pi}'\;\mathbf{1}^{\mathrm{T}} \;=\; 1 \end{cases} \tag{55}$$

The computation of the performance indices defined over GSPN models can be performed using the reward method discussed in Section 7.1 without any additional difficulty.

The advantage of solving the system by first identifying the REMC is twofold. First, the time and space complexity of the solution is reduced in most cases, since the iterative methods used to solve the system of linear equations tend to converge more slowly when applied with sparse matrices and an improvement is obtained by eliminating the vanishing states thus obtaining a denser matrix [29,15]. Second, by decreasing the impact of the size of the set of vanishing states on the complexity of the solution method, we are allowed a greater freedom in the explicit specification of the logical conditions of the original GSPN, making it easier to understand.

The method outlined in this section exploits the elegant mathematical structure of the problem to overcome the difficulties due to the presence of loops of immediate transitions. The loops considered in this derivation are of the "transient" type [29] and correspond to situations in which a steady-state analysis of the model is possible. The REMC is instead impossible to construct following this approach when the loop of immediate transitions is of the "absorbing" type so that during its evolution the net can be trapped into a situation from which it cannot exit. Except for very pathological cases [29] in which the model makes sense despite the presence of such absorbing loops, GSPNs of this type are considered non-well behaving and their analysis is stopped once the existence of absorbing loops of immediate transitions is discovered during the construction of the infinitesimal generator of the REMC.

Computational Considerations - The mathematically elegant solution techniques outlined in the previous part of this section suffer in practice of the difficulties due to the size of the CTMCs associated with these models and of the time-scale differences of the activities represented by transitions. In this subsection we will discuss the main difficulties encountered for the computation of the transient and steady-state probability distributions and we will outline some solution techniques; we will refer to the specialized literature for a deeper discussion of the problem.

Transient Solution - Uniformization Method

The first difficulty in the analysis of the CTMC associated with GSPN models comes from the evaluation of the transient probability distribution on the markings of the net.

The apparently simple solution of Equation (8.5)

$$\boldsymbol{\pi}'(\tau) \;=\; \boldsymbol{\pi}'(0)e^{\boldsymbol{Q}'\tau} \;=\; \boldsymbol{\pi}'(0)\sum_{k=0}^{\infty}\frac{(\boldsymbol{Q}'\tau)^k}{k!} \tag{56}$$

is unfortunately often rather difficult and unstable to compute [67]. Moler and Van Loan [19] discuss nineteen "dubious" ways to compute the exponential of

relatively small matrices whose accuracy heavily depends on the norms of the matrices.

One of the most commonly used methods for computing the transient probabilities and that avoids most of these problems is called the *uniformization* technique that is based on the following simple derivation.

Assume that the diagonal elements of the infinitesimal generator $\boldsymbol{Q}'$ are bounded, so that there exist a Γ such that:

$$|q'_{ii}| \leq \Gamma < \infty, \quad \forall \boldsymbol{m}_i \in TS \tag{57}$$

A (discretized) transition probability matrix $\boldsymbol{R}$ can be defined as follows

$$\boldsymbol{R} = \boldsymbol{I} + \frac{1}{\Gamma}\boldsymbol{Q}' \tag{58}$$

From this definition we have that $\boldsymbol{Q}' = \Gamma(\boldsymbol{R} - \boldsymbol{I})$, so that

$$\boldsymbol{\pi}'(\tau) = \boldsymbol{\pi}'(0)e^{\boldsymbol{Q}'\tau} = \boldsymbol{\pi}'(0)e^{\tau\Gamma\boldsymbol{R}-\tau\Gamma\boldsymbol{I}} = \boldsymbol{\pi}'(0)e^{-\tau\Gamma}e^{\tau\Gamma\boldsymbol{R}} \tag{59}$$

since $e^{-(\tau\Gamma)\boldsymbol{I}} = e^{-\tau\Gamma}\boldsymbol{I}$ and $\boldsymbol{R}$ and $\boldsymbol{I}$ commute. The transient distribution at time τ is thus obtained by computing an approximation to the infinite summation

$$\boldsymbol{\pi}'(\tau) = \boldsymbol{\pi}'(0) \sum_{k=0}^{\infty} \boldsymbol{R}^k e^{-\Gamma\tau} \frac{(\Gamma\tau)^k}{k!} \tag{60}$$

which is called the *uniformization equation* [67]. The advantages of this technique are the simplicity of its implementation and the possibility to control the approximation error. In fact, it is possible to easily identify the limit after which to truncate the series in order to reduce the resulting error below any predefined threshold ϵ [67]. The approximated result can thus be expressed in the following form:

$$\boldsymbol{\pi}'(\tau) = \boldsymbol{\pi}'(0) \sum_{k=0}^{K} \boldsymbol{R}^k e^{-\Gamma\tau} \frac{(\Gamma\tau)^k}{k!} \tag{61}$$

Steady-State Solution - Time Scale Decomposition

As we have seen at the beginning of this section, the steady state probability distribution of the markings of a GSPN model is obtained from the solution of a system of linear equations. This problem, that is mathematically simple and well understood, may pose considerable difficulties in the case of GSPNs due to the size of their reachability set and to the possibility of having within the infinitesimal generator of the corresponding CTMC rates that are several orders of magnitude different from each other. In this section we will not survey the many methods that can be employed for the solution of these large system of linear equations; the interested reader is referred to [67] for a comprehensive discussion of direct as well as of iterative techniques that can be employed to obtain the desired solution keeping under control the storage requirements as well as the computational costs. We will instead briefly outline a technique that

can be conveniently used to obtain approximate results when the GSPN models are of a special type and when transitions of different speeds are included in the net.

Consider the case of the operation of a simple processor/memory system in which the memory may fail while it is not accessed. Fig. 28 depicts the GSPN model of such a system in which failures happen quite rarely and repairs require a considerable amount of time to be completed. In a model of this type, besides the distinction between immediate and timed transitions, we can further classify timed transitions as *fast* and *slow*. In Fig. 28, transitions T_{req} and T_{str} can be considered fast, while transitions T_{fail} and T_{rep} can be assumed to be slow.

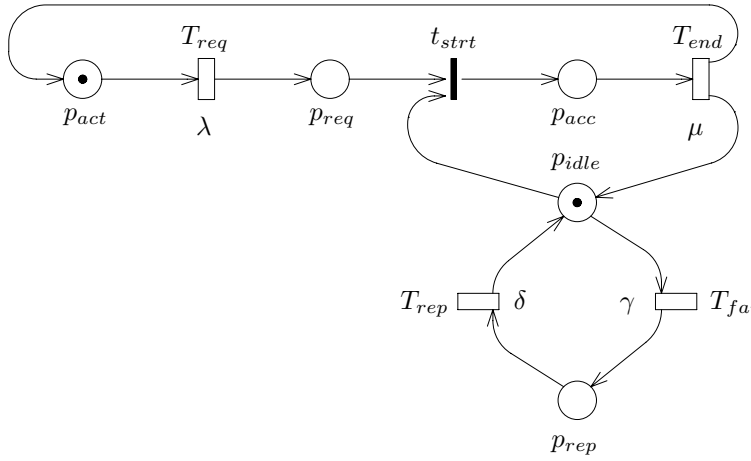


Fig. 28. A simple processor/memory system with failures

A technique for solving these models in a computationally convenient manner has been proposed by Ammar and Islam [11] and called Time Scale Decomposition Method (TSDM). This approach is based on the theory of *Near Complete Decomposability* due to Simon and Ando [66] and further investigated by Courtois [33]. In this case we may only observe that at the fast time scale, failures and repairs are so far apart in time, that during these intervals the system can be assumed to be fault-free. The theory of near decomposable systems says that as the process approaches the long-term equilibrium, a short-term equilibrium is reached between rare events, so that the short-term analysis can be approximately separated from the analysis of the long-run dynamics of such systems. The theory is developed by identifying within the Markov chains representing these systems, groups of states with (internal) strong interaction compared with the weak interaction existing among states of different groups. The method of

Ammar and Islam tries to identify these groups at the GSPN level by removing the slow transitions so that the net decomposes into several "fast" subnets that describe the short-run dynamics of the system. The individual solution of the CTMCs associated with these subnets provides the basis for the computation of aggregated representations that are used in the construction of a reduced subnet, called the aggregated GSPN (AGSPN), used for the analysis of the long-run dynamics of the system.

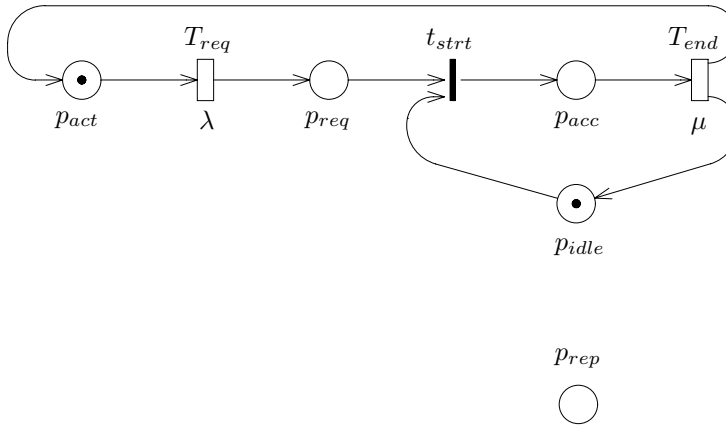


Fig. 29. Fast subnets derived form the simple processor/memory system with failures of Fig. 28

Following the technique proposed by Ammar and Islam, we remove the slow transitions from the net of Fig. 28 and we obtain the two subnets of Fig. 29, only one of which is interesting for the construction of the aggregated net of Fig. 30. Computing the solution of this first subnet (which is very simple since it has only two tangible markings corresponding to the processor active locally and to the processor accessing the memory), we observe that the firing rate of transition T_{a1} of the aggregated net is different from zero only when this fast submodels is in its first state.

Assuming that $\rho = \lambda/\mu$ and $\sigma = \gamma/\delta$, and indicating with $\boldsymbol{\pi}$ the solution of the original model of Fig. 28, with $\boldsymbol{\pi}^*$ its approximation computed with the TSDM of Ammar and Islam, with $\boldsymbol{\pi}^1$ the solution of the fast submodel of Fig. 29, and with P the solution of the aggregated model of Fig. 30, we have:

$$\pi^1(p_{act} + p_{idle}) \;=\; \frac{1}{1 \;+\; \rho}. \tag{62}$$

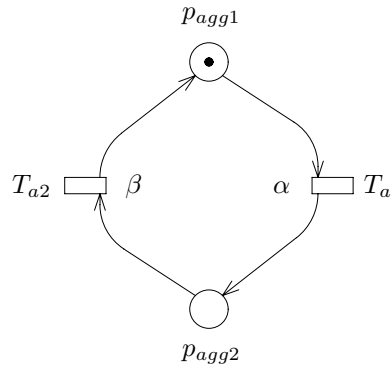


Fig. 30. Aggregated subnet derived form the simple processor/memory system with failures of Fig. 28

From the solution of the fast model, we have that the equivalent failure rate is:

$$\alpha \;=\; \frac{\gamma}{1 \;+\; \rho}, \tag{63}$$

while the repair rate remain unchanged:

$$\beta \;=\; \delta. \tag{64}$$

With these definitions, the solution of the aggregated (high level) model is easily obtained:

$$\mathcal{P}(p_{agg1}) \;=\; \frac{1 \;+\; \rho}{1 \;+\; \rho \;+\; \sigma}. \tag{65}$$

Once we know the probabilities of the aggregated states, we use them as conditional probabilities for obtaining detailed information on the states of the fast models. We thus obtain:

$$\pi^*(p_{act} + p_{idle}) \;=\; \frac{1}{1 \;+\; \rho \;+\; \sigma} \tag{66}$$

Solving the original (whole) model directly, we would have obtained:

$$\pi(p_{act} + p_{idle}) \;=\; \frac{1}{1 \;+\; \rho \;+\; \sigma \;+\; \rho\sigma\dfrac{\delta}{\lambda + \delta}}. \tag{67}$$

The difference between $\boldsymbol{\pi}$ and $\boldsymbol{\pi}^*$ is due to the approximate computation of the aggregated rate α of transition T_{a1} of the model of Fig. 29. The aggregated result would have been exact if the equivalent failure rate α had the following expression:

$$\alpha \;=\; \frac{\gamma}{1 \;+\; \rho\,(1 \;+\; \frac{\gamma}{\lambda+\delta})} \tag{68}$$

that differs from that of Equation (8.5) only for the ratio $\gamma/(\lambda+\delta)$ that obviously becomes negligible when $\gamma \ll \lambda$ as it is the case for our model.

In general the technique presented in [11] cannot be applied in such a straightforward manner and requires some clever intervention from the analyst, thus making it unsuitable for automatic applications. Under the condition of dealing with a somehow more restricted class of models, a variation of the above technique has been presented by Blakemore and Tripathi [16] that, working at the level of the state space of the CTMC underlying the GSPN model, first define the groups of states used for the aggregation, and then identify the transitions (of the GSPN model) that make the CTMC moving among these groups of states. Depending on the choice of the groups of states, the transitions that induce a change of state between groups (called *cross transitions*) can be either fast or slow. An implementation of this method should make sure that cross transitions are also slow in order for the TSD technique to be accurate, issuing a warning when this condition is not met. The interested reader will find the details of this last method, together with an algorithm for its automatic application, in [16].

8.6 Reducing GSPNs to SPNs

As we saw in the previous sections, immediate transitions, that have quite a beneficial impact on the modelling power of GSPNs, unfortunately make the overall analysis of GSPNs more complex than that of SPNs. Indeed, while for the latter the reachability graph is isomorphic to the state transition rate diagram of the CTMC underlying the SPN, such isomorphism does not exist for GSPNs, but is obtained only after the elimination of vanishing markings. Since loops of immediate transitions seldom appear in GSPN models, several solution methods have been devised that discard the vanishing states "on the fly" thus trading space with computational effort due to the fact that in some situations sequences of vanishing states are repeatedly generated to compute the transition rates among different pairs of tangible markings. Usually this method is computationally convenient, even if some extra work must be performed to preanalyze the model in order to discover the presence of situations that can not be treated with this technique. The choice of not saving vanishing markings has also the drawback of making certain performance indices related with the firing of immediate transitions impossible to compute. Details on the comparison of the advantages and disadvantages of storing vanishing markings can be found in [29]. An analysis procedure different from that described in the previous sections consists in the elimination of all immediate transitions from a given GSPN before starting the generation of its reachability set, so as to produce an SPN whose underlying continuous-time Markov chain is identical to that corresponding to the GSPN.

A complete and formal description of these reduction rules and of the class of GSPNs for which it is possible to compute an equivalent SPN, can be found in [22,34]. Here we only provide a concise and informal overview of the rules for the reduction of a GSPN model to an equivalent SPN in the case of free-choice conflicts.

The basic idea behind the elimination of immediate transitions is quite simple and can be easily explained in its natural form by means of an example. The model depicted in Fig. 31(a) is a free-choice GSPN subnet whose SPN counterpart is shown in Fig. 31(b). The two systems are equivalent as far as tangible states are concerned.

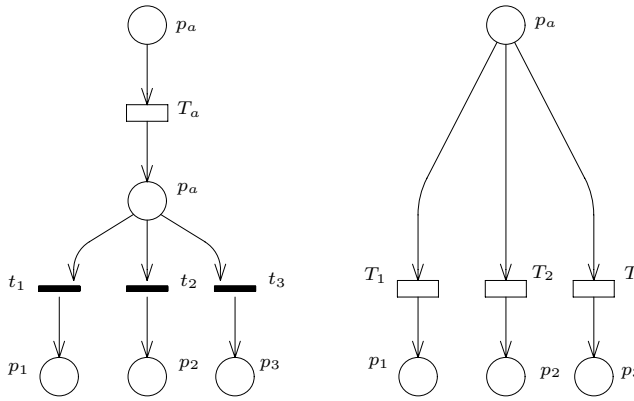


Fig. 31. Removing immediate transitions: the free-choice case

To understand how we can transform the GSPN into its corresponding SPN, consider what happens in the GSPN when T_a fires: the three immediate transitions t_1, t_2, and t_3 become enabled, due to the arrival of one token in place p_b; they are the only transitions that are enabled in the subnet, and the choice about which one to fire is made according to their associated weights. The basic behaviour of the subnet in Fig. 31(a) is therefore that of "moving" tokens from place p_a to one of the three places p_1, p_2, and p_3. The net in Fig. 31(b) is clearly equivalent to that of Fig. 31(a) from the point of view of the flow of tokens (token flow equivalence was defined in [13]).

When time is involved, token flow equivalence is not enough to ensure the possibility of freely interchanging the two types of models. In particular, the equivalence notion we are interested in must preserve the underlying stochastic behaviour of the net. We must therefore take into account not only the possible final states of the subnet, but also the rates at which the model transits from state to state.

The operations of the GSPN subnet induce a movement of a token from p_a to p_k, $k = 1, 2, 3$, with rate

$$\frac{w_a w_k}{\sum_{j=1,2,3} w_j} \tag{69}$$

where w_a is the rate of the exponential distribution associated with timed transition T_a, and w_k is the weight of immediate transition t_k, so that

$$\frac{w_k}{\sum_{j=1,2,3} w_j} \tag{70}$$

is the probability that t_k is the transition that actually fires when t_1, t_2, and t_3 are all enabled.

The timed transition T_k in the SPN model represents the firing of timed transition T_a followed by immediate transition t_k. If we define its firing rate as in Equation (69), then the rate at which the SPN subnet in Fig. 31(b) induces a movement of a token from p_a to p_k, given that transition T_a is enabled, is exactly the same as for the GSPN subnet, and therefore the rates of the underlying Markov processes are the same. Note that place p_b was deleted in the reduction process: this is not surprising because p_b is a vanishing place, a place that is never marked in a tangible marking.

The elimination procedure is somewhat more complex if the set of immediate transitions enabled by the firing of a timed transition does not form a free-choice conflict. Indeed, in this case the probability of firing an immediate transition may depend also on the other simultaneously enabled immediate transitions. We are thus forced to consider all possible cases. Details on this elimination procedure can be found in [22,34].

9 Conclusions

In this paper we have shown how Stochastic Petri Nets can be conveniently used for the analysis of complex models of DEDS and for their performance and reliability evaluation.

The advantage of net-based models, however, goes far beyond the modelling power of the formalism. In fact, the analysis of the structure of the graph and the computation of its algebraic properties provide information such as invariant conditions, flow-balance equations, and special structures of the reachability graph that can be used to identify models with peculiar solution characteristics, to optimize the solution techniques, and to develop approximation methods.

After an initial difficulty in accepting this new formalism due to its intrinsic complexity, SPNs are widely used today for the performance and reliability evaluation of many practical systems.

What was originally perceived as a disadvantage is now often appreciated because of the capability of the formalism to describe with precision complex phenomena that are typical of distributed and parallel systems. Moreover, the possibility of using the same model for both a qualitative analysis (functional validation) and an efficiency and reliability evaluation is understood as an important result in the assessment of real systems. It is a common belief that this approach should be further exploited and that more powerful and more user-friendly software tools must be developed to make this integrated study common practice for system designers.

This success also highlights a whole set of new problems since larger and larger models are being built and need to be analyzed. Dealing with large models is obviously difficult since even in the case of bounded nets, the size of their reachability sets can become enormous, making their numerical evaluation impossible and discrete-event simulation extremely expensive.

Many important results have been developed in the SPN field since the time of the introduction of this formalism. In our view, the most important ones are those using the net structure of the model to ease the modelling effort and to improve the efficiency of the solution methods. This allows the analyst to reason about the system at the net level while hiding the complexity of the underlying probabilistic structure.

We can summarize these new developments as follows:

- *General Distributions* - The assumption of the negative exponential distribution of firing times was soon felt to be too restrictive for a convenient representation of complex systems and several attempts were made to introduce general firing time distributions into the formalism. Firing delays that are characterized by phase-type distributions [57], were introduced by employing suitable subnet structures that could be easily embedded into GSPN models [20].
  Using the approach of identifying an embedded Markov chain, a technique has been proposed for the analysis of deterministic and stochastic Petri nets (DSPNs) [8]. In this case the embedded chain is used for the computation of the steady-state solution of nets in which at most one transition of constant delay is enabled in any marking. The transient analysis of DSPN models is presented in [27].
  It was soon recognized that when allowing full generality in the specification of the model the possibility for the computation of analytical or numerical solutions was lost and simulation was the only way to analyze these models [41,40].
  Recently, the class of DSPN models has been extended allowing the transitions to have generally distributed firing times, provided that the constraint of having at most one of these transitions enabled in each marking is still satisfied [39]. This class of models is also called Markov Regenerative SPNs [48,28] and special formulas for the computation of their steady-state solution were derived. A systematic study of this class of models can be found in [30].
- *Symmetries* - When dealing with complex systems, it often happens that their models can be constructed via the replication of many identical submodels. To deal with this problem, coloured Petri nets have been proposed to allow the construction of more compact representations [47].
  A special class of coloured Petri nets is that of *Stochastic Well Formed Petri Nets* (SWNs) in which restrictions are introduced on the functions that regulate transition firings and colour manipulations [36,25,24]. The important feature of SWNs is that the special form of their colour functions allows the direct construction of an aggregated state space. With this formalism the

symmetries intrinsic in the model are directly exploited to identify markings that are representative of large groups of states having similar characteristics. The aggregation method is fully automated and the direct generation of the aggregated Markov chain is obtained with considerable saving at the level of memory requirements. A significant advantage is also obtained when the reduced state space is still too large for the model to be solved numerically and we must resort to simulation to obtain the performance indices of interest. A method of symbolic simulation has been developed in which only the aggregated markings are generated [26].

- *Block Structure* - The problem of mastering the complexity of models of real systems has also been investigated using an approach driven by the idea that efficient solutions of difficult problems must be sought from the earliest stage of model construction. This approach tries to exploit the properties and the structure of the net to guide the solution method. This principle implies that, in order to understand how complex systems work, their behaviours must be considered as following from the composition of their individual parts. Thus in the development of the model, the analyst must maintain a local view of the different components avoiding the direct representation of global system features that, resulting from the interaction of the individual parts, could exhibit unexpected behaviours in pathological cases that are difficult to foresee. The Markov chain that underlies the entire model is only formally specified in this approach in terms of the Markov chains (transition probability matrices) of the individual submodels and of certain correcting factors that account for the interactions among the submodels. The complete transition probability matrix is never really constructed thereby allowing the solution of extremely large models in a very efficient manner. This solution technique also has the non-trivial advantage of being quite suitable for parallelization. In addition, the solution of the entire model is made easier when the submodels interact in very special ways [18], thus making the correcting factors extremely simple.
- *Product-Form SPNs* - To overcome the state-space explosion problem of the Markov chains associated with these models, a class of SPNs has been identified in which the steady-state probability distribution of their markings can be expressed in a product-form. The characterization of this class of SPNs was first expressed in terms of the special repetitive structures exhibited by their reachability graph [46,64] and subsequently in terms of structural criteria that can be easily checked by inspecting the incidence matrix of the net [43,42]. Proofs have been developed to show that it is possible to recognize whether SPNs have a product-form solution strictly from the results of their structural analysis. A complete characterization of this class of models can be found in [17].

Despite these results that we have briefly overviewed, the state-space explosion problem remains the major difficulty for using Petri net models in practical applications. There is general consensus that the only means of successfully dealing with large models is to use a "divide and conquer" approach in which the

solution of the entire model is constructed on the basis of the solutions of its individual submodels.

Compositionality is a property that is not natural in the Petri net field, but the tendency of constructing complex models from many small "building blocks" is becoming increasingly common and a need exists for also developing this approach for Petri nets. New theoretical results, mostly inspired by the research on "process algebras" [52] and "box calculus" [14], are being derived in this direction contributing to the development of a solid theoretical framework for compositionality both at the level of model construction and model solution.

Acknowledgement - The discussion contained in this work builds on the contents of several papers that the author has previously written on this topics in collaboration with several colleagues. It is thanks to them the extent to which the material contained in this report provides a good introduction to the use of Stochastic Petri Nets; any omissions and imprecisions, which are sure to exist, are instead the sole responsibility of the author. Special thanks go to Simona Bernardi for her help in harmonizing the notations and the figures throughout the whole report.

References

1. T. Agerwala. Putting Petri nets to work. *IEEE Computer*, pages 85–94, December 1979.
2. M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Transactions on Software Engineering*, 15(7):832–846, July 1989.
3. M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Generalized stochastic Petri nets revisited: Random switches and priorities. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, pages 44–53, Madison, WI, USA, August 1987. IEEE-CS Press.
4. M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.
5. M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, Cambridge, USA, 1986.
6. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley, 1995.
7. M. Ajmone Marsan, G. Balbo, and K.S. Trivedi, editors. *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.
8. M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponential transition firing times. In *Proc. 7^{th} European Workshop on Application and Theory of Petri Nets*, pages 151–165, Oxford, England, June 1986.
9. H. Alaiwan and G. Memmi. Algorithmes de recherche des solutions entieres positives d'un systeme d'equations lineaires homogeneus en nombres entieres. *Revue Technique Thomson-CSF*, 14(1):125–135, March 1982. in French.
10. H. Alaiwan and J. M. Toudic. Research des semiflots, des verrous et des trappes dans le reseaux de Petri. *Technique et Science Informatiques*, 4(1), February 1985.

11. H.H. Ammar and S.M. Rezaul Islam. Time scale decomposition of a class of generalized stochastic Petri net models. *IEEE Transactions on Software Engineering*, 15(6):809–820, June 1989.
12. G. Balbo, M. Silva, and editors. *Performance Models for Discrete Event Systems with Synchronizations: Formalisms and Analysis Techniques*. KRONOS, Zaragoza, Spain, 1998.
13. G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 359–376. Springer Verlag, Bad Honnef, West Germany, February 1987.
14. E. Best, R. Devillers, and J. Hall. The Petri box calculus: a new causal algebra with multilabel communication. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 609 of *LNCS*, pages 21–69. Springer Verlag, 1992.
15. A. Blakemore. The cost of eliminating vanishing markings from generalized stochastic Petri nets. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.
16. A. Blakemore and S.K. Tripathi. Automated time scale decomposition and analysis of stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 248–257, Toulouse, France, October 1993. IEEE-CS Press.
17. R.J. Boucherie. A characterization of independence for competing Markov chains with applications to stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 117–126, Toulouse, France, October 1993. IEEE-CS Press.
18. P. Buchholz. Hierarchical high level Petri nets for complex system analysis. In *Proc.* 15^{th} *Intern. Conference on Applications and Theory of Petri Nets*, number 185 in LNCS, Zaragoza, Spain, 1994. Springer-Verlag.
19. Moler C. and C. Van Loan. Nineteen doubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978.
20. P. Chen, S.C. Bruell, and G. Balbo. Alternative methods for incorporating non-exponential distributions into stochastic Petri nets. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, pages 187–197, Kyoto, Japan, December 1989. IEEE-CS Press.
21. G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, February 1993.
22. G. Chiola, S. Donatelli, and G. Franceschinis. GSPN versus SPN: what is the actual role of immediate transitions? In *Proc.* 4^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 20–31, Melbourne, Australia, December 1991. IEEE-CS Press.
23. G. Chiola, S. Donatelli, and G. Franceschinis. Priorities, inhibitor arcs, and concurrency in P/T nets. In *Proc.* 12^{th} *Intern. Conference on Application and Theory of Petri Nets*, Aarhus, Denmark, June 1991.
24. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.
25. G. Chiola and G. Franceschinis. Colored GSPN models and automatic symmetry detection. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

26. G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation*, 24, 1995. Special Issues on Performance Modelling Tools.
27. H. Choi, G. Kulkarni, and K.S. Trivedi. Transient analysis of deterministic and stochastic petri nets. In *Proc.* 14^{th} *Intern. Conference on Application and Theory of Petri Nets*, Chicago, Illinois, June 1993. Springer Verlag.
28. H. Choi, V.S. Kulkarni, and K.S. Trivedi. Markov regenerative stochastic Petri nets. In *Proc. of Performance 93*, Rome, Italy, September 1993.
29. G. Ciardo. *Analysis of large Petri net models.* PhD thesis, Department of Computer Science, Duke University, Durham, NC, USA, 1989. Ph. D. Thesis.
30. G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 170–179, Toulouse, France, October 1993. IEEE-CS Press.
31. G. Ciardo and C. Lindemann. Analysis of deterministic and stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 160–169, Toulouse, France, October 1993. IEEE-CS Press.
32. J.E. Coolhan and N. Roussopoulos. Timing requirements for time-driven systems using augmented Petri nets. In *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.
33. P.J. Courtois. *Decomposability: Queueing and Computer Systems Applications.* Academic Press, New York, 1977.
34. S. Donatelli. *L'uso delle reti di Petri per la valutazione e la validazione di sistemi di grandi dimensioni.* PhD thesis, Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy, February 1990. (in italian).
35. J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. Extended stochastic Petri nets: Applications and analysis. In *Proc. PERFORMANCE '84*, Paris, France, December 1984.
36. C. Dutheillet and S. Haddad. Aggregation and disaggregation of states in colored stochastic Petri nets: Application to a multiprocessor architecture. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.
37. G. Florin and S. Natkin. Les reseaux de Petri stochastiques. *Technique et Science Informatiques*, 4(1), February 1985.
38. G. Florin and S. Natkin. Matrix product form solution for closed synchronized queueing networks. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, pages 29–39, Kyoto, Japan, December 1989. IEEE-CS Press.
39. R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. In *Proc. of Performance 93*, Rome, Italy, September 1993.
40. P. J. Haas and G. S. Shedler. Regenerative stochastic Petri nets. *Performance Evaluation*, 6(3):189–204, September 1986.
41. P.J. Haas and G.S. Shedler. Regenerative simulation of stochastic Petri nets. In *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.
42. W. Henderson and D. Lucic. Exact results in the aggregation and disaggregation of stochastic Petri nets. In *Proc.* 4^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 166–175, Melbourne, Australia, December 1991. IEEE-CS Press.
43. W. Henderson and P.G. Taylor. Aggregation methods in exact performance analysis of stochastic Petri nets. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, pages 12–18, Kyoto, Japan, December 1989. IEEE-CS Press.

44. M.A. Holliday and M.K. Vernon. A generalized timed Petri net model for performance analysis. In *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.
45. K. Lautenbach. Linear algebraic technique for place/transition nets. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 142–167. Springer Verlag, Bad Honnef, West Germany, February 1987.
46. A.A. Lazar and T.G. Robertazzi. Markovian Petri net protocols with product form solution. *Performance Evaluation*, 12:67–77, 1991.
47. C. Lin and D.C. Marinescu. On stochastic high level Petri nets. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, Madison, WI, USA, August 1987. IEEE-CS Press.
48. V. Mainkar, H. Choi, and K. Trivedi. Sensitivity analysis of Markov regenerative stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.
49. J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In *Proc.* 2^{nd} *European Workshop on Application and Theory of Petri Nets*, Bad Honnef, West Germany, September 1981. Springer Verlag.
50. P. M. Merlin and D. J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, September 1976.
51. J. F. Meyer, A. Movaghar, and W. H. Sanders. Stochastic activity networks: Structure, behavior, and application. In *Proc. Intern. Workshop on Timed Petri Nets*, pages 106–115, Torino,Italy, July 1985.
52. R. Milner. *Communication and concurrency*. Prentice Hall, 1989.
53. M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, UCLA, Los Angeles, CA, 1981. Ph.D. Thesis.
54. M.K. Molloy, T. Murata, and M.K. Vernon, editors. *Proc. Intern. Workshop on Petri Nets and Performance Models*, Madison, Wisconsin, August 1987. IEEE-CS Press.
55. T. Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
56. S. Natkin. *Les Reseaux de Petri Stochastiques et leur Application a l'Evaluation des Systemes Informatiques*. PhD thesis, CNAM, Paris, France, 1980. These de Docteur Ingegneur.
57. M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, Baltimore, MD, 1981.
58. J. D. Noe and G. J. Nutt. Macro e-nets representation of parallel systems. *IEEE Transactions on Computers*, 31(9):718–727, August 1973.
59. J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
60. C.A. Petri. Communication with automata. Technical Report RADC-TR-65-377, Rome Air Dev. Center, New York, NY, 1966.
61. C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Transactions on Software Engineering*, 6(5):440–449, September 1980.
62. C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Cambridge, MA, 1974. Ph.D. Thesis.
63. W. Reisig. *Petri Nets: an Introduction*. Springer Verlag, 1985.
64. T.G. Robertazzi. *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Springer Verlag, 1991.

65. J. Sifakis. Petri nets for performance evaluation. In H. Beilner and E. Gelenbe, editors, *Proc.* 3^{rd} *Intern. Symp. IFIP*, pages 75–93, 1978.
66. H.A. Simon and A. Ando. Aggregation of variables in dynamic systems. *Econometrica*, 29, 1961.
67. W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, New Jersey, USA, 1994.
68. F. J. W. Symons. Introduction to numerical Petri nets, a general graphical model of concurrent processing systems. *Australian Telecommunications Research*, 14(1):28–33, January 1980.
69. R.S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1962.
70. C.Y. Wong, T.S. Dillon, and K.E. Forward. Timed places Petri nets with stochastic representation of place time. In *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.
71. W.M. Zuberek. Timed Petri nets and preliminary performance evaluation. In *Proc.* 7^{th} *Annual Symposium on Computer Architecture*, pages 88–96, La Baule, France, May 1980.

Non-Markovian Analysis

Reinhard German

Computer Science 7 (Computer Networks and Communication Systems)
Friedrich-Alexander Universität Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
german@informatik.uni-erlangen.de
http://www7.informatik.uni-erlangen.de/~german/

Abstract. If in stochastic modeling the idealized assumption of exponential distributions is removed, the resulting stochastic process is non-Markovian. In this tutorial paper we give an overview of possible analytic approaches for such non-Markovian models. The modeling framework of stochastic Petri nets is used, but the ideas are applicable to other frameworks as well, if a state space can be constructed. We give a detailed presentation of one analysis approach which is based on the method of supplementary variables and give a brief review of another analysis approach which is based on embedding. A model of a timer for holding a connection is used as a tutorial example and a model for a medium access mechanism in wireless networks is used as a more complex example.

1 Introduction

The exponential distribution is popular in model-based performance and dependability evaluations. Due to its memoryless property, the stochastic process underlying a stochastic model is a continuous-time Markov chain which can be analyzed easily. However, there is some debate whether the exponential distribution is indeed adequate.

For arrival processes we know two properties which give us confidence in the exponential distribution. In many cases the superposition of a large number of independent renewal processes tends to a Poisson process, which has exponentially distributed interarrival times. Furthermore, if a fixed interval is taken, the arrival instants of a Poisson process inside that interval are completely random. However, for service times there is no such reasoning.

Consider a simple non-Markovian model, a queuing system with a Poisson arrival process, generally distributed service times, and infinite capacity (an M/G/1 system). The well-known Pollaczek-Khintchine formula tells us that the mean waiting time W for a customer in the queue and service unit is given by

$$W = \bar{X} + \frac{\rho \bar{X}(1 + C_X^2)}{2(1 - \rho)},$$

where $\bar{X}$ is the mean service time, C_X^2 the squared coefficient of variation of the service time, and ρ the utilization (ratio of arrival rate and service rate). If we

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 156–182, 2001.

would ignore the general service time and would just replace it by an exponential distribution with identical mean, the squared coefficient of variation would be equal to one ($C_X^2 = 1$). For a utilization $\rho = 0.5$ and mean service time $\bar{X} = 1$ the mean waiting time for exponential service thus gets $W = 2$. Contrast this with the situation when $C_X^2 = 100$, the mean waiting time is then $W = 51.5$, approximately 25 times larger!

According to our experience it is hard to predict whether the higher moments of a distribution influence the measures of interest in a stochastic model significantly. There are of course cases in which only the mean value affects the steady-state result, this phenomenon is referred to as *insensitivity*. If steady-state is sufficient for the problem at hand, one can proceed in those cases with the analytically tractable exponential distribution. There are also cases in which the higher moments affect the results, but the difference is negligible compared with the other approximations of the model. Compare for instance in the above example the mean waiting time for exponential services ($W = 2$) with the mean waiting time for deterministic service times ($W = 1.5$). The results are of the same order of magnitude, the difference is probably not significant taking into account that a queue is an abstract model of a real system. From our experience we can derive the following rules of thumb: for measures like mean waiting times or throughputs, differences become often significant if the variances of some distributions get large; for measures like loss probabilities even going from the exponential distribution to deterministic values may cause differences in several orders of magnitude.

The following list gives some examples of modeling problems in which non-exponential distributions are natural:

- If measurements are taken (e.g., of some service time, such as transmissions of packets in a communication system), it is very unlikely that the empirical distribution fits well to an exponential distribution. (Very often packet lengths shows three peeks: small, medium and large packets.) The empirical distribution can be fed into in a model.
- If only the minimum and the maximum of some quantity is known and more information is not available, the uniform distribution would be a good choice.
- The Weibull distribution is common in reliability, since it can have age-dependent failure rates and is thus better suited to model parts of the so-called bath-tub curve.
- Many measurements of various quantities in computer and communication systems, such as file transmission times in the Internet and file sizes on a host give evidence of heavy tails. This means that the complement of the distribution function does not decrease exponentially fast (as it is the case for the exponential and similar distributions), but do decrease according to a polynomial law. One example of such a heavy-tailed distribution is the Pareto distribution with suitable parameters.

However, some quantities in a stochastic model are deterministic and not random:

- The clock cycles in computers are fixed. Very often activities last fixed multiples of these clocks cycles. For instance, packet switches of a wide area network have an internal clock for their switching fabric and an external clock with which packets are switched.
- For given bitrates and fixed packet lengths, transmission times are fixed. This situation arises for instance in ATM cell switching or often in IP routing where the default segment size is 536 bytes. Also, many communication systems have a slotted frame structure: frames are composed of a fixed number of slots with fixed size.
- Deadlines in real-time systems or timeouts in communication systems are fixed.
- In reliability models repair times and scheduled maintenance intervals have often a fixed length.

Both lists are non-exhaustive.

Assuming that this is sufficient motivation for non-Markovian models, we provide a review of analytic approaches for such models in the rest of the paper. We concentrate on state-space-based analysis, that is, for the analysis the state space of the model needs to be constructed. Recently, most work in this direction has been performed in the context of *stochastic Petri nets* (SPNs). We also use this framework for the presentation. We refer the reader to [1] in this volume for a tutorial about SPNs and do not discuss their definition and semantics here. The reader should note that all presented analysis approaches would be applicable to other specification techniques as well. If for instance process algebras or state charts are extended in a similar way as Petri nets to stochastic Petri nets, the same analysis is possible.

In Sec. 2 we give a short discussion of possible analytic approaches. One of these approaches, the *method of supplementary variables* is discussed in detail in Sec. 3. Another approach, which is based on *embedding* is briefly reviewed in Sec. 4. Tool support is discussed in Sec. 5 and an application example is given in Sec. 6.

This tutorial paper is based on the textbook [16]. In some parts of this paper we give a slower introduction than in the textbook and in some parts we summarize the results of the textbook. The interested reader is also directed to [33] as a standard reference for Markovian SPNs, material can also be found in [38] and [22]. In [31] the stationary analysis of deterministic and stochastic Petri nets (DSPNs) is also investigated. DSPNs are a restricted class of non-Markovian models in which both exponentially timed activities and deterministically timed activities are allowed. Another tutorial paper about the development in non-Markovian SPNs is [3].

2 Analysis Approaches for Non-Markovian Models

In a time-continuous Markovian model all distributions are exponential (geometric in case of a time-discrete model). Due to the memoryless property it is

sufficient to consider just the discrete state space of the underlying continuous-time Markov chain (CTMC) (discrete-time Markov chain (DTMC)) [23]. From the theory of stochastic processes, queuing theory, reliability theory, and from the work on SPNs a number of analysis approaches is known for non-Markovian stochastic models as well. In the following we give a brief review of these approaches.

State-space expansion based on *continuous phase-type distributions* [35,36]. A non-exponential distribution is given by the time to absorption in a CTMC. As a prominent example, deterministic times can be approximated by an Erlang distribution, which is composed of a sequence of exponential phases. Theoretically, an infinite number of phases is required to achieve a deterministic time, the same is true for distributions with a discrete part. By the introduction of phases an underlying CTMC with an expanded state space can be constructed. The state space grows by a multiplicative factor of order M^N, where N is the number of concurrent non-exponential activities and M is the number of phases. In the context of SPNs, the concept was introduced in [10]. Some relief in the state-space increase is possible by taking symmetries into account using lumping and/or using compositional techniques based on Kronecker algebra [21].

The *method of supplementary variables* [19,15]. In this method the memory of activities with non-exponential distribution is represented as a "supplementary variable" and added to the state space. This leads to a hybrid state space: a component for the discrete state and a component for the continuous supplementary variable. It is then possible to form equations which describe the dynamics of the system just by using first principles and then to analyze this system of equations. If each discrete state is supplemented by at most one continuous variable, efficient analysis procedures are possible. In the stationary case the solution can be based on the analysis of Markov chains only and no discretization is required. In the transient case the analysis can be based on a one-dimensional discretization of the continuous variable. It is in principle possible to use the method of supplementary variables also in more general cases with more than one continuous variable added to each state, but then the numerical analysis becomes more tedious.

The method of supplementary variables comes in several flavours: the supplementary variables can be age or rest variables (expressing the time since start or until end of an activity), forward or backward equations can be formulated, and the equations can be expressed as differential or integral equations. Age variables can be thought of as clocks, whereas rest variables can be thought of as decrementing timers. Note that rest variables correspond very much to event scheduling used in discrete-event simulation. In any case, supplementary variables provide a straightforward way to extend formal methods by time.

In Sec. 3 we will use age variables and describe the dynamics of the stochastic process by forward differential equations. Another variant is based on *generalized semi-Markov processes* (GSMPs), which is applied for the analysis of DSPNs with concurrent deterministically timed activities [32,31]. In the GSMP approach rest variables are used and the dynamics of the stochastic process is described at

equidistant time instants by integral equations. The solution requires a kernel matrix with entries which can depend on $2N$ continuous variables which all have to be discretized.

Probably the best-known method is based on *embedding* (also known as *Markov renewal theory*). In this method the time axis is considered at certain time instants, at which certain events occur. These time instants are referred to as regeneration points, and it is possible to define a DTMC, referred to as *embedded Markov chain* at these time instants. Then a generalized Markov renewal equation can be formulated to express the evolution over time. The method requires roughly the same restriction as the method of supplementary variables (only one memory can be present in each state). Some generalizations are possible and will be discussed in Sec. 4.

State-space expansion based on *discrete phase-type distributions* [7,26]. In this approach the greatest common divisor of all deterministic firing times has to be taken as the basic time step. However, the approach requires the encoding of phase counters in the markings and it leads to a significant increase of the state space (e.g., when the deterministic times differ just slightly or by several orders of magnitude). Kronecker algebra may be used in the discrete case as well [40]. As a further generalization of modeling power it is possible to mix discrete and continuous phase type distributions in the same model [28].

3 Analysis with Supplementary Variables

Non-Markovian SPNs are used for the model description. Instead of giving a formal definition here, we just say that the model class is the same as *generalized stochastic Petri nets* (GSPNs), as defined in [1] in this volume or in [33], in which the firing times of the timed transitions are also allowed to have a general distribution. We will discuss some subtle aspects of the underlying semantics in Sec. 3.2. However, we will assume throughout the paper that the transitions with non-exponential firing time distribution are mutually exclusive, that is, they must not be concurrent. As a consequence, at most one age variable has to be added to each state.

Figure 1 gives an overview of the required analysis steps. The model is described as an SPN together with the measures of interest. Reward expressions provide a convenient and flexible way to formulate the measures *on the same level as the model*.

The SPN model is mapped on the reduced reachability graph (RRG) which contains only tangible markings. The measures are mapped on corresponding state-space based representations. The RRG is then used to construct the stochastic process consisting of discrete states and continuous supplementary variables. The data structures for the representation of the stochastic process are three sparse matrices, as we will see later.

The numerical analysis is executed on these data structures and yields two vectors: the state probabilities and the firing frequencies. These two vectors and

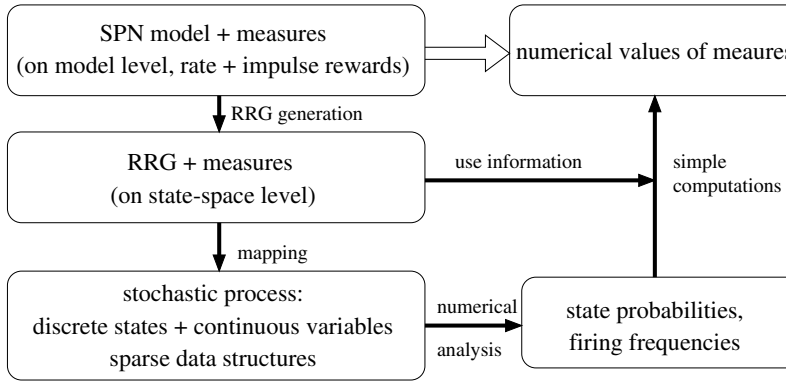


Fig. 1. Overview of the required analysis steps

the state-space based representation of the measures are used to compute the numerical values of the measures in comparably simple computations.

3.1 Construction of the Underlying Stochastic Process

The RRG is constructed as for GSPNs, for instance by a breadth-first search over the tangible markings and elimination of the immediate markings on-the-fly. Typically, the construction of the RRG and of the underlying stochastic process is performed in one pass. We present in the following an example in which we construct the stochastic process with its discrete states and continuous supplementary variables.

Figure 2 shows an SPN model which models the timeout for a connection in a communication system, referred to a "on-demand connection with delayed release" (OCDR). The model is an adapted version of [24] and taken from [16]. In the OCDR mechanism a connection is setup when the first packet arrives and the connection is held when the buffer is empty until a timeout expires. We assume that packets arrive according to a Poisson process with a rate of 100 per second, that the packet lengths are uniformly distributed from 100 to 2000 bytes, that the bit rate is 10 Mbps, that connection setup time is 10 ms and the timeout is 20 ms. One second is taken as the underlying time unit. Transition `arrival` models packet arrivals and has an exponentially distributed firing time with rate $\lambda = 100$. Transition `service` models packet transmissions and has a uniformly distributed firing time from 0.00008 to 0.0016. Transition `connect` models connection setup and has a deterministic firing time of 0.01 and transition `release` models the timeout and has a deterministic firing time of 0.02. Transition `start` models the start of a packet transmission and is immediate.

Measures of interest could be the mean number of packets in the system $N = E[\#buffer + \#busy]$ and the system throughput $S = E[\#service]$. $E[\cdot]$ is the expected value operator, $\#place$ gives the number of tokens in *place* (referred

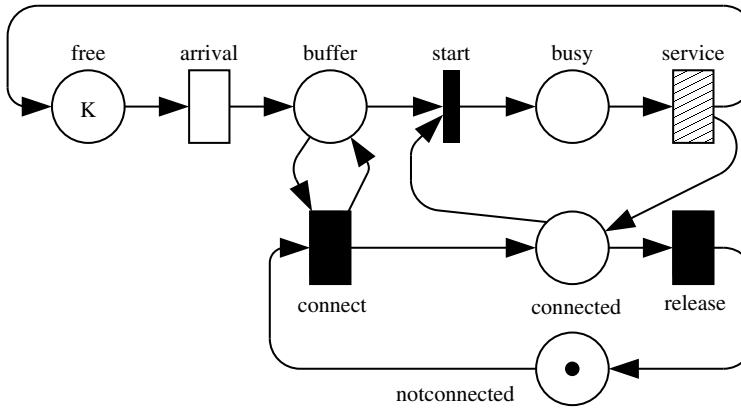


Fig. 2. Example SPN model

to as *rate reward*), and *#transition* gives a value equal to one when *transition* fires (referred to as *impulse reward*). Formally, an impulse reward is a Dirac impulse with area one, see [16], pp. 47–51. These are two examples of reward-based measure definitions. More complex expressions can be formed by common arithmetic and boolean operators. Note that the formulation is in terms of model-level entities (and not in terms of the state space).

We will now construct the underlying stochastic process. Formally, it is a tuple

$$\{(N(t), X(t)), t > 0\},$$

where $N(t)$ is the discrete process which describes the tangible marking and $X(t)$ is the continuous supplementary variable which describes the age (that is, the time since the instant of enabling) of the enabled transition with non-exponential distribution. For convenience we use the following numbering for the states of the discrete process:

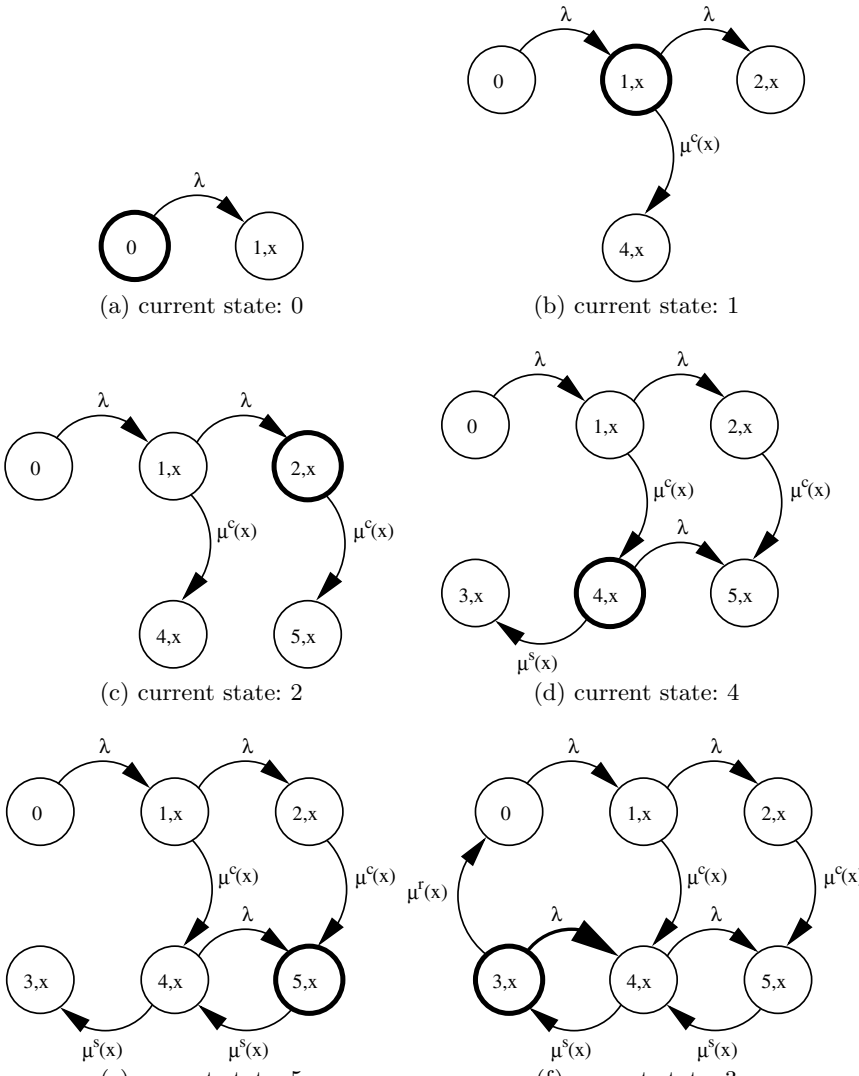
$$N(t) = \#buffer + \#busy + 1_{\{\#notconnected=0\}}(K+1).$$

Table 1 shows the mapping of state numbers on tangible markings for $K = 2$ (the ordering of places is: `free`, `buffer`, `busy`, `connected`, `notconnected`).

Figure 3 (a) - (f) shows the single steps in the construction of the stochastic process. The stochastic process is visualized by its state-transition diagram. Starting with state 0 (the initial marking shown in Fig. 2), each step explores the enabled transitions and next reachable states of the current state. The current state is drawn with a thick line. If in a state only exponentially timed transitions are enabled, no age variable has to be added to the discrete state. This is the case for state 0. An exponential state transition is labelled by its rate. For instance, the firing of `arrival` in the initial marking corresponds to the arc from state 0 to state 1 labelled by λ. If in a state a non-exponentially timed

Table 1. State numbers of the OCDR model

state number	tangible marking
0	(2,0,0,0,1)
1	(1,1,0,0,1)
2	(0,2,0,0,1)
3	(2,0,0,1,0)
4	(1,0,1,0,0)
5	(0,1,1,0,0)



(a) current state: 0 (b) current state: 1

(c) current state: 2 (d) current state: 4

(e) current state: 5 (f) current state: 3

Fig. 3. Construction of the underlying stochastic process

transition is enabled, an age variable x is added to the state description. This is the case for all states besides state 0. x represents the time since enabling of the non-exponentially timed transition which is enabled in this state. Therefore x is the age variable of `connect` in states 1 and 2, of `service` in state 4 and 5, and of `release` in state 3. A state transition caused by a non-exponentially timed transition is labelled by its age-dependent firing rate. For instance, the firing of `connect` in state 1 corresponds to the arc from state 1 to state 4 labelled by

$$\mu^c(x) = \frac{f^c(x)}{1 - F^c(x)},$$

where $F^c(x)$ and $f^c(x)$ are the distribution and density functions of the firing time of `connect`. Analogously, $\mu^s(x)$ and $\mu^r(x)$ are the age-dependent firing rates of `service` and `release`.

The exponential transition from state 3 to state 4 in Fig. 3 (f) requires special attention. It corresponds to the firing of `arrival` when `buffer` and `busy` are empty but `connected` holds a token. In state 3 deterministic transition `release` is also enabled (modeling the timeout). After `arrival` has fired immediate transition `start` removes the token from `connected` and `release` is preempted (timeout timer is stopped). After some time `release` will be enabled again. It is therefore important to specify exactly what happens with the age variable in this situation. In this model we assume that it is reset to zero when `release` becomes preempted. This preemption policy is referred to as *preemptive repeat different*.

The model-level measure definitions can be translated into rate and impulse reward vectors. Rate reward vectors are later multiplied with the vector of state probabilities and impulse reward vectors with the vector of general firing frequencies (see below) to get the desired expected values. In the given example we get for N the rate reward vector $\mathbf{rr}_N = (0, 1, 2, 0, 1, 2)$ and we get for S the impulse reward vector $\mathbf{ir}_S = (0, 0, 0, 0, 1, 1)$.

3.2 Model Class and Notation

The class of stochastic processes under consideration is given by the marking processes of SPNs with the following properties:

1. The set of tangible markings (the discrete state space) is finite.
2. The general transitions are mutually exclusive (there is no tangible marking in which two or more are enabled).
3. The preemption policy of all transitions is preemptive repeat different.
4. The general firing time distributions are marking-independent.

Properties 2 and 3 ensure that at most one supplementary variable has to be added to each discrete state. Generalizations of properties 3 and 4 are possible and discussed in Sec. 3.5.

For describing the stochastic process we introduce some notation:

- t is the model time and x is the supplementary age variable,
- $g \in G$ is a general transition, G the set of all general transitions,

- $F^g(x)$, $f^g(x)$, $\mu^g(x)$, and $\bar{F}^g(x)$ refer to the distribution, density, instantaneous rate, and complementary distribution function, respectively,
- $\mathcal{S}$ is the state space,
- $\mathcal{S}^g$, $\mathcal{S}^G$, and $\mathcal{S}^E$ are the subsets in which g is enabled, any general transition is enabled, only exponential transitions are enabled, respectively.

In the example of Sec. 3.1 the three general transitions `service`, `connect`, and `release` are abbreviated by their first letters s, c, and r, respectively. The general firing time distributions are denoted as $F^s(x)$, $F^c(x)$, and $F^r(x)$. The set of general transitions is given by $G = \{s, c, r\}$. According to the given numbering of states we have for $K = 2$ the following state space and subsets:

- $\mathcal{S} = \{0, 1, 2, 3, 4, 5\}$
- $\mathcal{S}^E = \{0\}$, $\mathcal{S}^G = \{1, 2, 3, 4, 5\}$
- $\mathcal{S}^s = \{4, 5\}$, $\mathcal{S}^c = \{1, 2\}$, and $\mathcal{S}^r = \{3\}$

The state transitions of the stochastic process are given by the three matrices $\mathbf{Q}$, $\bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ defined as follows:

- The non-diagonal entry q_{ij}, $i \neq j$, is the rate of the exponential state transitions from i to j which do not preempt a general transition; the diagonal entry q_{ii} is the negative sum of all rates of exponential state transitions out of state i (including those which preempt a general transition).
- $\bar{q}_{ij}$ is the rate of the exponential state transitions from i to j which preempt a general transition.
- $\delta_{ij} = Pr\{N(t^+) = j \mid N(t^-) = i, \text{a general transition fires at } t\}$; in words: δ_{ij} is the *branching probability*, it is the probability that the firing of a general transition g leads to state j, given that g fires in i.

All matrices are square matrices and their dimension is given by the cardinality of the state space, $|\mathcal{S}|$. $\mathbf{Q}$ is a *defective generator matrix* since the outgoing rates of preemptive state transitions are not included.

Figure 4 shows the matrices in the example of Sec. 3.1. The left side shows the matrices and the right side shows the state-transition diagram with only those state transitions which are represented by the matrix entries.

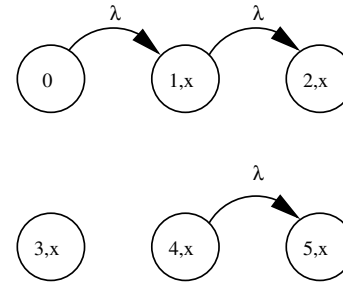
For the analysis of the stochastic process we need quantities defined in Table 2; $\boldsymbol{\pi}(t)$, $\boldsymbol{\varphi}(t)$, $\boldsymbol{\pi}(t, x)$, and $\mathbf{p}(t, x)$ are vectors of these quantities and have dimension $|\mathcal{S}|$. $\pi_n(t, x)\, dx$ can be interpreted as the probability that the discrete state is n and that the elapsed general firing time is in a neighbourhood of x. $p_n(t, x)\, dx$ can be interpreted as the probability that the discrete state is n and that the elapsed general firing time is in a neighbourhood of x, given that the transition has not yet fired. For fixed values of t, $\pi_n(t, x)$ is a defective density function and $p_n(t, x)$ a defective instantaneous rate function. $\varphi_n(t)$ is the rate of firing of the general transition enabled in state n at time t.

We also use the following *filter concept*: for subset $\mathcal{S}^G \subseteq \mathcal{S}$ a *filter matrix* $\mathbf{I}^G$ is defined as the diagonal matrix whose ith entry is equal to one if $i \in \mathcal{S}^G$ and equal to zero otherwise. In a similar way, filters can be defined for $\mathcal{S}^g$ and $\mathcal{S}^E$. Filters are used to filter out certain parts of the vectors and matrices. Given

Table 2. Definition of quantities required for the analysis

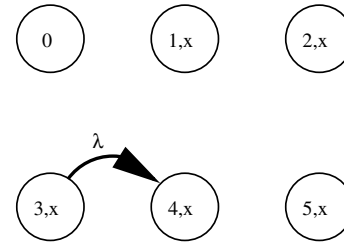
quantity	definition
state probability	$\pi_n(t) = Pr\{N(t) = n\}$
age density	$\pi_n(t,x) = \frac{\partial}{\partial x} Pr\{N(t) = n, X(t) \le x\}$
age intensity	$p_n(t,x) = \pi_n(t,x)/\bar{F}^g(x)$
general firing frequency	$\varphi_n(t) = \int_0^\infty \pi_n(t,x)\mu^g(x)\,dx$

$$\mathbf{Q} = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & 0 & 0 \\ 0 & -\lambda & \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & -\lambda & \lambda \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



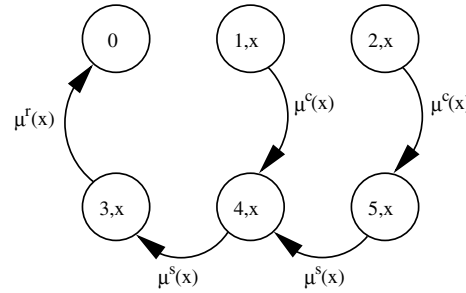
(a) matrix $\mathbf{Q}$ and corresponding state transitions

$$\bar{\mathbf{Q}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



(b) matrix $\bar{\mathbf{Q}}$ and corresponding state transition

$$\mathbf{\Delta} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



(c) matrix $\mathbf{\Delta}$ and corresponding state transitions

Fig. 4. Matrices and corresponding state transitions

a vector $\mathbf{v}$ and a filter $\mathbf{I}^A$, the filtered vector is $\mathbf{v}^A = \mathbf{v}\mathbf{I}^A$, given a matrix $\mathbf{M}$ the filtered matrix is $\mathbf{M}^A = \mathbf{I}^A\mathbf{M}$. It is possible to filter out both rows and columns of a matrix: a multiplication with a filter from the left filters out rows and a multiplication from the right filters out columns. Applied to a matrix the first superscript denotes the set of originating states and the second superscript the set of destination states. The concept of filtering allows us to write all vector/matrix equations without caring about the cardinality of the different portions of $\mathcal{S}$ and about the ordering of the states.

3.3 Transient Analysis

The dynamics of the stochastic process can be described by a system of differential and integral equations. As illustration we derive a single state equation for state 1 of the example of Sec. 3.1.

Consider the possible change of the system in an interval $[t, t + \Delta t]$ and assume that $x > 0$ at the beginning of the interval. The age density of state 1 can change by the following single-step state transitions: state 1 can be left, either to state 2 or 4; an entry from state 0 in one step is not possible since $x > 0$. This is expressed as a difference equation:

$$\pi_1(t + \Delta t, x + \Delta t) = \pi_1(t, x) - \pi_1(t, x)(\lambda \Delta t + \mu^c(x)\Delta t) + o(\Delta t),$$

where $o(\Delta t)$ is a term with higher order than Δt and represents state transition paths with more than one step. Subtracting $\pi_1(t, x)$ on both sides and dividing by Δt leads to:

$$\frac{\pi_1(t + \Delta t, x + \Delta t) - \pi_1(t, x)}{\Delta t} = -(\lambda + \mu^c(x))\pi_1(t, x) + \frac{o(\Delta t)}{\Delta t},$$

taking the limit $\Delta t \to 0$ leads then to the *partial differential equation* (PDE):

$$\frac{\partial}{\partial t}\pi_1(t, x) + \frac{\partial}{\partial x}\pi_1(t, x) = -(\lambda + \mu^c(x))\pi_1(t, x).$$

Substituting $\pi_1(t, x) = p_1(t, x)\bar{F}(x)$ in the PDE we can derive the simplified PDE

$$\frac{\partial}{\partial t}p_1(t, x) + \frac{\partial}{\partial x}p_1(t, x) = -\lambda p_1(t, x),$$

which no longer contains a term with $\mu^c(x)$.

Now consider the possible change of the system in an interval $[t, t + \Delta t]$ and assume that $x = 0$ at the beginning of the interval. The system can not have been in state 1 immediately before t, a single-step entry is only possible from state 0:

$$Pr\{N(t) = 1, X(t) \leq \Delta t\} = \int_0^{\Delta t} \pi_1(t, x)\, dx = \pi_0(t)\lambda \Delta t + o(\Delta t).$$

Expanding the integral up to the first term, taking $\Delta t \to 0$, and substituting $\pi_1(t,x) = p_1(t,x)\bar{F}(x)$ leads to

$$p_1(t,0) = \lambda \pi_0(t),$$

which is a *boundary condition* for the PDE.

Equations for the other states can be derived with similar reasoning, for all states with a supplementary variable we get a PDE and a boundary condition, for all states without a supplementary variable we get an *ordinary differential equation* (ODE). These equations must be combined with initial conditions for $t = 0$ and integrals which express the state probabilities and general firing frequencies in terms of the age intensities.

Using the matrix notation of Sec. 3.2 the complete system of equations is given in Box 1. A detailed derivation can be found in [16], Chapter 7.

ODEs:

$$\frac{d}{dt}\boldsymbol{\pi}^E(t) = \boldsymbol{\pi}^E(t)\mathbf{Q}^{E,E} + \boldsymbol{\varphi}(t)\boldsymbol{\Delta}^{G,E} + \boldsymbol{\pi}^G(t)\bar{\mathbf{Q}}^{G,E}$$

PDEs:

$$\frac{\partial}{\partial t}\mathbf{p}^G(t,x) + \frac{\partial}{\partial x}\mathbf{p}^G(t,x) = \mathbf{p}^G(t,x)\mathbf{Q}^G$$

Initial conditions:

$$\mathbf{p}^G(0,x) = \boldsymbol{\pi}^G(0)\delta(x)$$

Boundary conditions:

$$\mathbf{p}^G(t,0) = \boldsymbol{\pi}^E(t)\mathbf{Q}^{E,G} + \boldsymbol{\varphi}(t)\boldsymbol{\Delta}^{G,G} + \boldsymbol{\pi}^G(t)\bar{\mathbf{Q}}^{G,G}$$

Integrals:

$$\boldsymbol{\varphi}(t) = \sum_{g\in G}\int_0^\infty \mathbf{p}^g(t,x)f^g(x)\,dx, \quad \boldsymbol{\pi}^G(t) = \sum_{g\in G}\int_0^\infty \mathbf{p}^g(t,x)\bar{F}^g(x)\,dx$$

Box 1: Transient state equations

The PDEs describe a propagation of the age intensities along characteristic lines with slope one (the lines on which t and x increase with same speed), they can therefore be simplified to:

$$\mathbf{p}^G(t,x) = \mathbf{p}^G(t_0+\vartheta, x_0+\vartheta) = \mathbf{p}^G(t_0,x_0)e^{\mathbf{Q}^G\vartheta}. \tag{1}$$

A numerical analysis of the equation system is possible by discretization. In [16] an algorithm is described which steps over discretized values of t and needs to store at the current value of t the age densities for discrete values of x. Therefore, only discretized values in one dimension need to be stored. The main operation of the algorithm is to evaluate the age densities at $(t+\vartheta, x+\vartheta)$ from known values at (t,x). Figure 5 shows the computational dependencies

between the age densities at different points. From Eq. (1) we see that these computations are similar to the transient evaluation of a Markov chain; it is possible to use uniformization for it. These computations are responsible for the major computational costs of the algorithm. The other steps of the algorithm use the information of the remaining equations (the ODEs, boundary conditions, and integrals); these steps are Runge-Kutta ODE solution, integration, inter- and extrapolation. The algorithm uses an adaptive step-size, two error control mechanisms and is able to deal with possible discontinuities explicitly. It has been realized in the tools SPNica and TimeNET, we have solved models up to approximately 100,000 states with it.

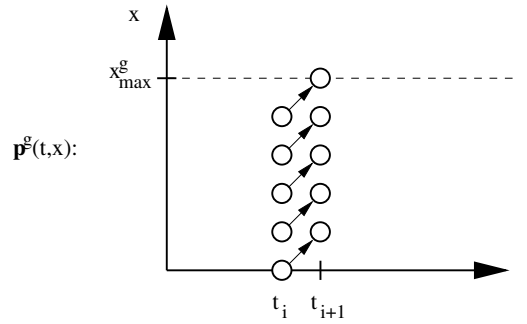


Fig. 5. Discretization of the PDE system

3.4 Stationary Analysis

If "long-term average" measures are sufficient, the state equations and their analysis can be simplified such that no discretization is required. Very often in performance related studies such long-term results are sufficient whereas in reliability evaluations mostly time-dependent results are required.

The long run corresponds to taking t to infinity: in the equations t cancels out and we arrive at a system of ODEs combined with some linear equations (i.e., $\pi_n = \lim_{t\to\infty} \pi_n(t)$). As a technical problem sometimes this limit (referred to as steady-state) does not exist (think of a model cycling between two states with deterministically timed state transitions). In such a case we can take time-averaged limits (i.e., $\pi_n = \lim_{t\to\infty} \frac{1}{t} \int_0^t \pi_n(\tau)\, d\tau$), leading to the same equations, referred to as the stationary state equations.

With this limit operation we get the vectors $\boldsymbol{\pi}$, $\boldsymbol{\varphi}$, $\boldsymbol{\pi}(x)$, and $\mathbf{p}(x)$ and the stationary state equations shown in Box 2.

Simple operations transform this system to a form suitable for analysis. The ODEs solution can be expressed as a matrix exponential and inserted into the integrals. If we define the two matrices $\boldsymbol{\Omega}$ and $\boldsymbol{\Psi}$

The ODEs become balance equations:

$$\mathbf{0} = \boldsymbol{\pi}^E \mathbf{Q}^{E,E} + \boldsymbol{\varphi} \boldsymbol{\Delta}^{G,E} + \boldsymbol{\pi}^G \bar{\mathbf{Q}}^{G,E}$$

The PDEs become ODEs:

$$\frac{d}{dx}\mathbf{p}^G(x) = \mathbf{p}^G(x)\mathbf{Q}^G$$

Boundary conditions:

$$\mathbf{p}^G(0) = \boldsymbol{\pi}^E \mathbf{Q}^{E,G} + \boldsymbol{\varphi} \boldsymbol{\Delta}^{G,G} + \boldsymbol{\pi}^G \bar{\mathbf{Q}}^{G,G}$$

Integrals:

$$\boldsymbol{\varphi} = \sum_{g \in G} \int_0^\infty \mathbf{p}^g(x) f^g(x)\, dx, \quad \boldsymbol{\pi}^G = \sum_{g \in G} \int_0^\infty \mathbf{p}^g(x) \bar{F}^g(x)\, dx$$

Additional normalization condition:

$$\boldsymbol{\pi}\mathbf{e} = 1$$

Box 2: Stationary state equations

$$\boldsymbol{\Omega} = \sum_{g \in G} \mathbf{I}^g \int_0^\infty e^{\mathbf{Q}^g x} f^g(x)\, dx, \;\; \boldsymbol{\Psi} = \sum_{g \in G} \mathbf{I}^g \int_0^\infty e^{\mathbf{Q}^g x} \bar{F}^g(x)\, dx, \tag{2}$$

the firing frequencies $\boldsymbol{\varphi}$ and general state probabilities $\boldsymbol{\pi}^G$ get:

$$\boldsymbol{\varphi} = \sum_{g \in G} \int_0^\infty \mathbf{p}^g(0) e^{\mathbf{Q}^g x} f^g(x)\, dx = \mathbf{p}^G(0)\boldsymbol{\Omega}$$

and

$$\boldsymbol{\pi}^G = \sum_{g \in G} \int_0^\infty \mathbf{p}^g(0) e^{\mathbf{Q}^g x} \bar{F}^g(x)\, dx = \mathbf{p}^G(0)\boldsymbol{\Psi}.$$

Insertion of this into the balance equations and boundary conditions and adding them leads to the linear equations

$$\mathbf{0} = \mathbf{w}\mathbf{S},$$

where $\mathbf{w} = \boldsymbol{\pi}^E + \mathbf{p}^G(0)$ and $\mathbf{S} = \mathbf{Q}^E + \boldsymbol{\Omega}\boldsymbol{\Delta} + \boldsymbol{\Psi}\bar{\mathbf{Q}} - \mathbf{I}^G$. Insertion into the normalization condition leads to

$$\boldsymbol{\pi} = \mathbf{w}\mathbf{D},$$

where $\mathbf{D} = \mathbf{I}^E + \boldsymbol{\Psi}$. In [16] Chapter 8 a more detailed derivation can be found.

It can be shown that $\mathbf{S}$ is the generator of a continuous time Markov chain (to which we refer as *embedded CTMC*) and that the stationary solution is unique if this embedded CTMC has at most one recurrent class. The required algorithmic steps are summarized as follows:

1. compute the integrals of the matrix exponentials of $\mathbf{\Omega}$ and $\mathbf{\Psi}$, a generalized version of uniformization can be used for this purpose,
2. solve the linear system of the embedded CTMC $\mathbf{0} = \mathbf{wS}$, subject to the normalization condition $\boldsymbol{\pi} = \mathbf{wD}$ by common methods for Markov chains,
3. back substitute $\mathbf{w}$ in order to get the state probabilities and general firing frequencies: $\boldsymbol{\pi} = \mathbf{wD}$, $\boldsymbol{\varphi} = \mathbf{w\Omega}$.

From $\boldsymbol{\pi}$ and $\boldsymbol{\varphi}$ it is easy to compute the values of the measures by multiplying with the rate and impulse reward vectors. In the example of Sec. 3.1 the mean waiting time of packets is given by the ratio of N and S (following Little's law): $W = (0,1,2,0,1,2)\boldsymbol{\pi}/(0,0,0,0,1,1)\boldsymbol{\varphi}$. With the TimeNET implementation of the algorithm and with $K = 1000$ we get the curve for the mean waiting time shown in Fig. 6. The computation of all points of the curve typically lasts some minutes on a "normal" workstation.

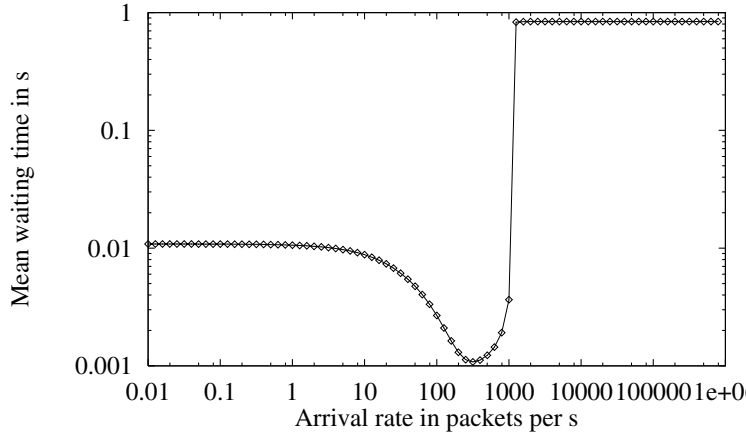


Fig. 6. Mean waiting time versus packet arrival rate for the OCDR model

3.5 Generalizations

A number of generalizations of the model class are possible [16]. Instead of "preemptive repeat different" also the policy "preemptive resume" is possible, where the age of a preempted transition is preserved until it gets enabled again (the transient case has been covered in [44]). This is useful for modeling activities which can be interrupted but which do not loose their performed work in that period. Additionally it is possible to let the general distributions of the firing times depend on the current marking. This is useful for modeling activities with varying speeds, as for instance transmission of a large message over a link with varying transmission rates. Finally, it is possible to deal with reducible models

in the stationary case with the same costs as the presented stationary analysis algorithm for irreducible models, see [16], Chapter 11. This is useful in some reliability models, for instance it is possible to compute the mean time to absorption or the probability to end up in safe/unsafe failure states.

4 Analysis Based on Embedding

Another approach for analyzing non-Markovian models is based on embedding. More generally speaking, it find its roots in Markov renewal theory. Most textbooks about queueing systems present this approach for M/G/1 and related systems, e.g., [29]. Historically, this approach was used to derive the first results for non-Markovian SPNs [34].

The idea is to sample the process at certain instants of time referred to as *regeneration points*. By an appropriate choice the process is memoryless at these regeneration points and its evolution becomes a probabilistic replica after each regeneration point. It is then possible to define a discrete-time Markov chain, referred to as the *embedded Markov chain* (EMC), to study the dynamics of this EMC, and to derive from its solution the solution of the actual process.

In case of a non-Markovian SPN with the properties given at the beginning of Sec. 3.2, the regeneration points can be chosen as follows:

- If the process is in an exponential state $n \in \mathcal{S}^E$, the instant of the next state transition is the next regeneration point.
- If the process is in a general state $n \in \mathcal{S}^G$, the instant when the general transition fires or is preempted is the next regeneration point.

The situation is illustrated in Fig. 7: in general states exponential state transitions are possible and have to be taken into account.

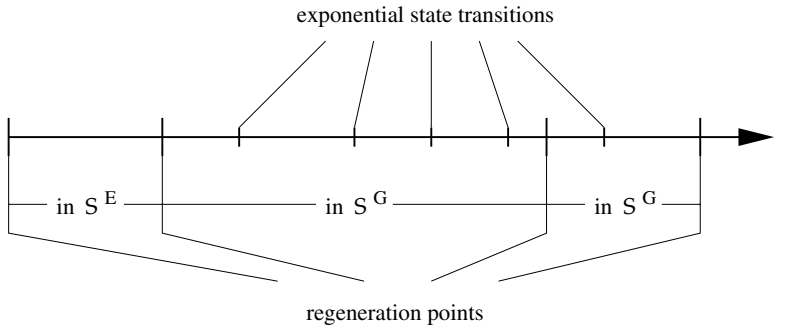


Fig. 7. Regeneration points in a non-Markovian SPN

The time-dependent behaviour can be expressed by the so-called *generalized Markov renewal equations* [6], a system of Volterra integral equations of the

second type. An analysis is possible by discretization. As a major difference to the method of supplementary variables a matrix and not just a vector of unknown functions has to be discretized. A Laplace domain analysis is also possible but restricted to models with small state space.

Stationary state equations can be derived from the generalized Markov renewal equations by taking t to infinity or directly from the regenerative structure of the process. The EMC expresses the state transition probabilities from one regeneration point to the other, it is represented by a stochastic matrix $\mathbf{P}$

$$\mathbf{P} = \mathbf{I}^E - \mathrm{diag}^{-1}\left(\mathbf{Q}^E\right)\mathbf{Q}^E + \mathbf{\Omega\Delta} + \mathbf{\Psi}\bar{\mathbf{Q}},$$

where $\mathrm{diag}^{-1}(\cdot)$ is the inverse of the diagonal matrix restricted to non-zero entries. The first two terms on the right side represent the transition probabilities in exponential states, the third term represents the transition probabilities according to the firing of general transitions and the last term represents transition probabilities according to the preemption of general transitions. The stationary solution of this EMC can be obtained from the solution of the linear system

$$\mathbf{u} = \mathbf{uP},\ \mathbf{ue} = 1.$$

The stationary probability of state n of the actual process is then found by dividing the expected sojourn time in state n by the expected time between two arbitrary regeneration points. This operation is expressed in matrix terms as:

$$\boldsymbol{\pi} = \frac{\mathbf{uC}}{\mathbf{uCe}}.$$

Summarizing, the required algorithmic steps are:

1. compute the integrals of the matrix exponentials of $\mathbf{\Omega}$ and $\mathbf{\Psi}$,
2. solve the linear system of the EMC $\mathbf{u} = \mathbf{uP}$, subject to the normalization condition $\mathbf{ue} = 1$,
3. back substitute $\mathbf{u}$ in order to get the state probabilities: $\boldsymbol{\pi} = \mathbf{uC}/\mathbf{uCe}$.

The similarities with the algorithm derived with supplementary variables are obvious. It is indeed possible to formally transform the state equations derived with supplementary variables and those with Markov renewal theory into each other [20]. In the stationary case, both algorithms can be regarded as identical, but in the transient case there are significant differences.

Two extensions of the approach based on Markov renewal theory are possible. The algorithm can be organized in a pure iterative version which does not require the storage of $\mathbf{\Omega}$ and $\mathbf{\Psi}$ [17]. Storage of these matrices is expensive since they are densely populated. In another work the embedding scheme has been extended to cascaded intervals (referred to a *cascaded embedding*) [14]. This gives a limited support for concurrent deterministic transitions. A layered iterative algorithm can be used for the stationary analysis and phase counters or discretization are not required. More work has also been done for dealing with different preemption policies [4,2,42,41,43] and for dealing with simultaneously enabled general transitions [37].

5 Tool Support

The following is a list of tools which support the analysis of non-Markovian models.

- **ESP** was developed at the University of Turin by Cumani and Bobbio [10]. It is based on Fortran and has a textual interface. Generally distributed firing times as well as preemption policies preemptive repeat different and preemptive resume can be represented by continuous phase type distributions. The expanded state space is constructed automatically.
- **DSPNexpress** was developed by Lindemann and coworkers at the Technische Universität Berlin and University of Dortmund [30,31]. It is especially tailored to the analysis of DSPNs, the analysis components are based on embedding and GSMPs. DSPNexpress provides a graphical user interface (GUI).
  See http://www4.cs.uni-dortmund.de/home/lindemann/
- **UltraSAN** was developed by Sanders and coworkers at the University of Arizona and at the University of Illinois at Urbana-Champaign [9,39]. Besides various analysis and simulation components, hierarchical modeling features, and a GUI, UltraSAN provides a component for the stationary analysis of DSPN-like models based on embedding.
  See http://www.crhc.uiuc.edu/PERFORM/
- **TimeNET** was developed as a successor of DSPNexpress at the Technische Universität Berlin by the author and coworkers [18,12]. It provides components for the transient and stationary analysis and simulation for non-Markovian SPNs, modeling with discrete-time SPNs, colored and hierarchical SPNs, a GUI, animation of the token game and on-line result visualization. The analysis components for non-Markovian SPNs are based on the approaches described in this paper. Figure 8 gives a snapshot of the GUI with the running tutorial example.
  See http://pdv.cs.tu-berlin.de/forschung/timenet/
- **WebSPN** was developed by Bobbio, Puliafito, Scarpa, and Telek from the Universities of Turin, Catania, and Budapest. It provides a Web-accessible GUI and is aimed for the analysis of non-Markovian SPNs with different preemption policies. The analysis component relies on discrete-time phase type distributions and on automatic mapping on an expanded DTMC [26].
  See http://sun195.iit.unict.it/ webspn/webspn2/
- **TANGRAM-II** was developed at the Federal University of Rio de Janeiro by de Souza e Silva and coworkers [5]. It provides an object-oriented modeling paradigm for the networking context. It includes models with exponential and deterministic activities and offers a stationary analysis component based on embedding and on optimized iteration formulas.
- **SPNica** was developed by the author as a prototype tool in Mathematica. It offers a realization of several analysis algorithms for non-Markovian SPNs. It takes advantage of many features of Mathematica (formulation of expressions, visualization, ...) but is not designed for efficiency. It is possible to solve

models with up to approximately 1000 states and to inspect the internal data structures. Down-loadable from ftp://ftp.wiley.co.uk/pub/books/german

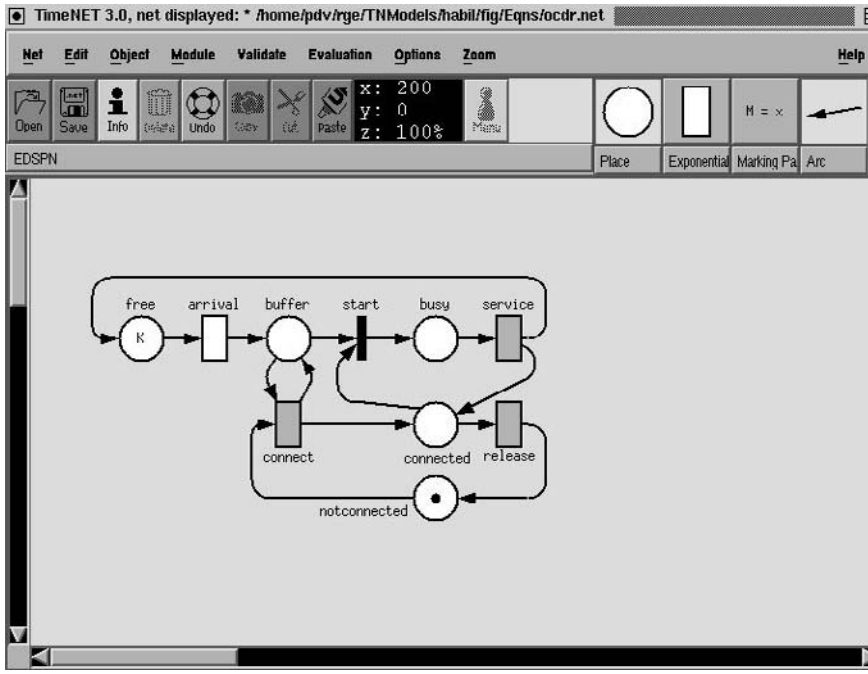


Fig. 8. Graphical user interface of TimeNET 3.0

A list of Petri net tools is accessible at the Petri net Web site via http://www.daimi.au.dk/PetriNets/tools/. Recently, a number of tools have been developed which have an open architecture and support multiple modeling formalisms: SMART [8] at the College of William and Mary, Möbius [11] at the University of Illinois at Urbana-Champaign, and IDEAS at Duke University [13]. An open architecture allows the exchange of evaluation components with other tools, and multiple modeling formalisms allow a user to select a domain-specific modeling formalism.

6 Example: IEEE 802.11 WLAN MAC

The tutorial example we have been using so far is relatively simple. In this section we outline a modeling study with a more complex model to give the reader a better impression of the potential use of the methodology. The results are obtained with TimeNET and SPNica. The system under study is the medium

access control (MAC) for wireless local area networks (WLANs) according to IEEE 802.11 [27]. The MAC mechanism is a variant of so-called carrier sense multiple access, a collision detection like in wired media is not practical. We compare two proposed mechanisms: *Basis Access* (BA) and *RTS/CTS* (RTS). The interesting modeling question is to determine the performance (in terms of throughput and mean delay) of the two mechanisms in the presence of collisions (transmissions which start at almost the same time get garbled and waste bandwidth).

A detailed model of the mechanism (the two variants can be modeled by different firing times of the transitions) is shown in Fig. 9. It models one station with a Poisson arrival process (exponential transition `gen`). Successful transmissions are modeled by general transition `Ttxsucc`, colliding transmissions by general transition `Ttxcoll` and the backoff by general transition `Tbackoff`. The whole WLAN with N stations is modeled by duplicating and connecting the shown subnet N times (TimeNET provides mechanisms which makes this duplication easier). The detailed model tries to model the internal behaviour of the mechanism as correctly as possible. However, it is not possible to analyse this model numerically, not only because of the concurrent general transitions, also because the state space gets too large. Therefore we used discrete-event simulation to get numerical values out of this model (with 99 % confidence intervals and maximum relative error of 1 %).

From the detailed model the compact model shown in Fig. 10 can be derived by folding the station subnets and by introducing some approximations. Transition `gen` is still exponential but has now "infinite-server" semantics (the firing rate is multiplied with the number of tokens in place `idle`). Deterministic transition `Tvuln` models the "vulnerable period" in which stations do not percept an ongoing transmission and deterministic transition `Ttimeout` models the timeout in which a station waits for an acknowledgement. General transition `Ttxsucc` and `Ttxcoll` have the same meaning as before. The backoff is now approximated by exponential transition `Tbackoff`. This non-Markovian SPN can be analyzed by the method of supplementary variables or by embedding, as described in this paper.

Simulation results from the detailed model and analysis results from the compact model are now presented. We compare Basic Access and RTS/CTS. Figure 11 shows the throughput vs. the load and Figure 12 shows the mean waiting time to transmit a packet vs. the throughput. We see that the analytic results from the non-Markovian model do very well approximate the simulation results. From a system point of view, Basic Access shows throughput decrease and instabilities for high loads.

More information about the MAC mechanism, the models, and the results can be found in [16], Chapter 15. In [25] the model is extended by other mechanisms of the standard and the modeling of the backoff counter is improved.

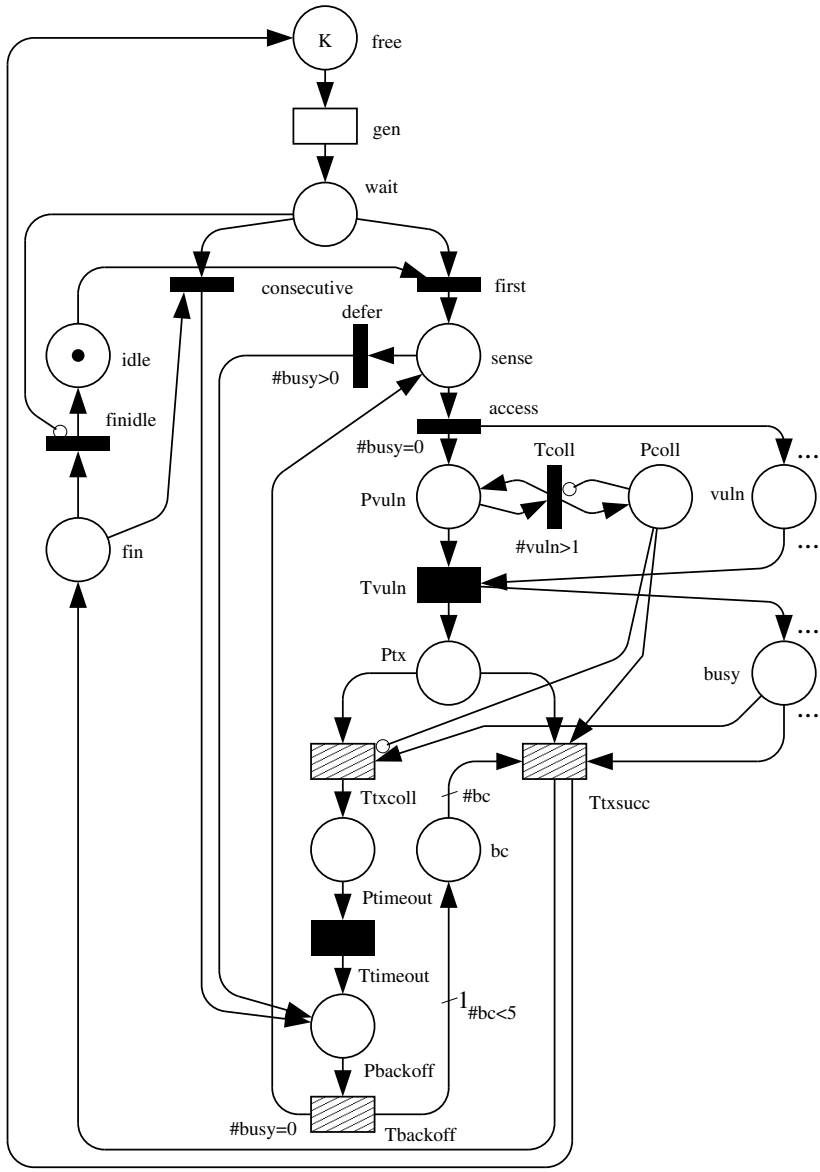


Fig. 9. Detailed SPN model of WLAN MAC

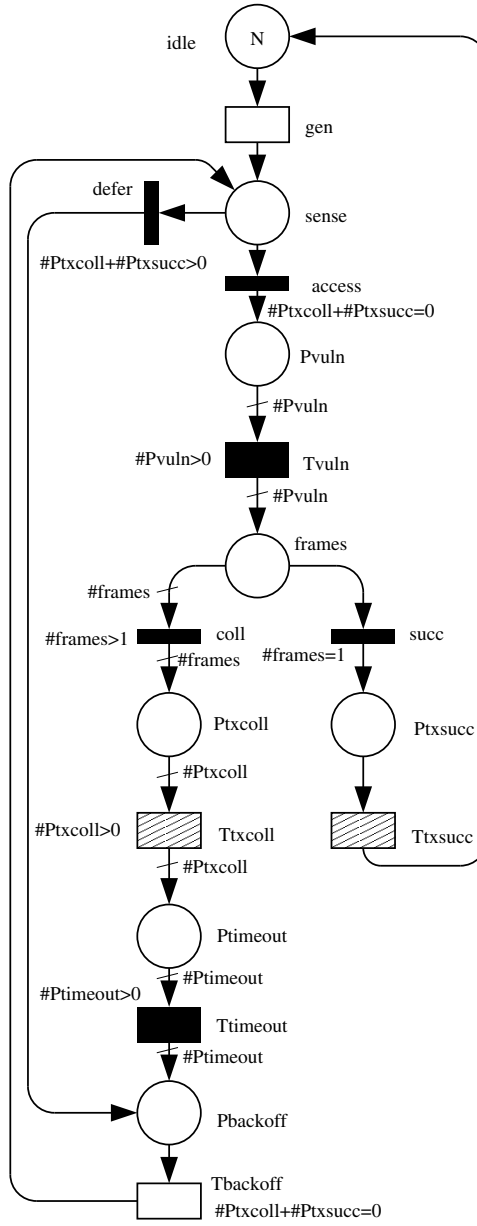


Fig. 10. Compact SPN model of WLAN MAC

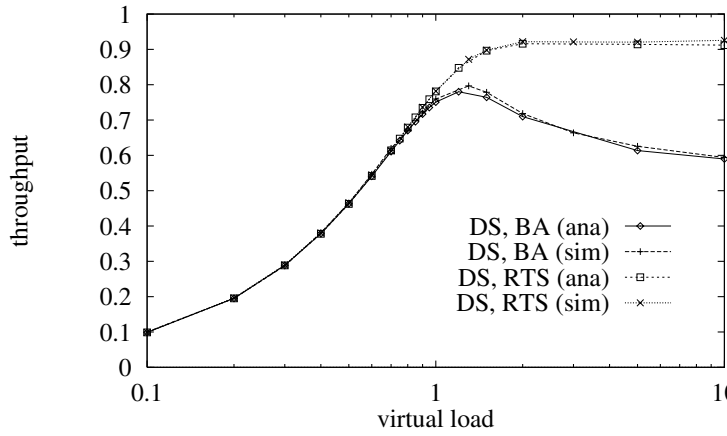


Fig. 11. DSSS: throughput vs. virtual load

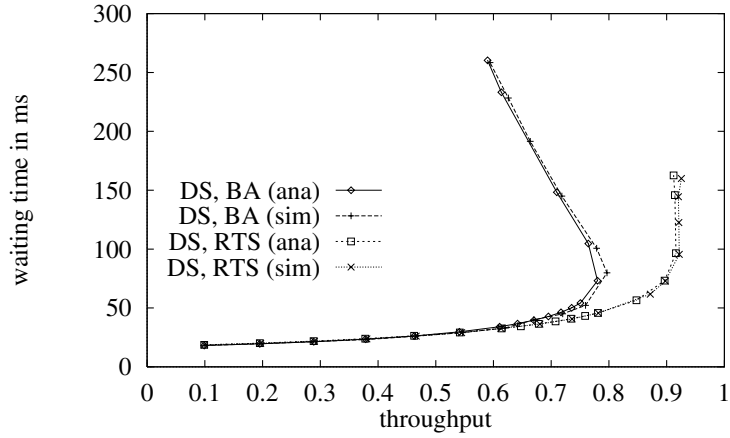


Fig. 12. DSSS: mean waiting time vs. throughput

7 Conclusions

The main obstacles for the numerical analysis of stochastic models are the largeness of the state space, the kind in which general distributions appear, and, not covered in this paper, the stiffness. Stiffness occurs when the model parameters differ in several orders of magnitude. In this case, the "conventional" analytic methods either get slow or can even give wrong results.

The analysis of non-Markovian models has matured for cases in which the generally timed activities are mutually exclusive. Here it is possible to realize stationary analysis methods which do not require discretization or phase-type expansion, maybe the major problem remains the fill-in of some needed matrices. Transient analysis methods can be formulated which do require only a one-dimensional discretization.

If non-exponential activities can happen concurrently, one either can use phase-type expansion (continuous, discrete, or mixed), use the GSMP approach (but discretizations are required), use the cascaded embedding approach if applicable, or resort to discrete-event simulation. If analytic methods are available they are of course preferable. This is especially the case if the interesting measures have small values, known as *rare events* in simulation. However, if analysis gets too costly (in terms of time for implementing the analysis algorithm or in terms of storage and timing requirements of the actual computations), we believe it is more efficient to use simulation at some point.

Finally, we repeat our statement that the presented analysis methods are not restricted to Petri nets. It is possible to extend other modeling frameworks in a similar way and to apply the same approaches.

References

1. G. Balbo. Introduction to stochastic Petri nets. This volume.
2. A. Bobbio, V.G. Kulkarni, A. Puliafito, M. Telek, and K. Trivedi. Preemptive repeat identical transitions in Markov regenerative stochastic Petri nets. In *Proc. 6th Int. Conf. on Petri Nets and Performance Models*, pages 113–122, Durham, NC, 1995.
3. A. Bobbio, A. Puliafito, M. Telek, and K. S. Trivedi. Recent developments in non-Markovian stochastic Petri nets. *Journal of Systems Circuits and Computers*, 8(1):119–158, 1998.
4. A. Bobbio and M. Telek. Markov regenerative SPN with non-overlapping activity cycles. In *Proc. Int. Performance and Dependability Symp.*, pages 124–133, Erlangen, Germany, 1995.
5. R. M. L. R. Carmo, L. R. de Carvalho, E. de Souza e Silva, M. C. Diniz, and R. R. Muntz. Performance/availability modeling with the TANGRAM-II modeling environment. *Perf. Eval.*, 33:45–65, 1998.
6. H. Choi, V. G. Kulkarni, and K. S. Trivedi. Markov regenerative stochastic Petri nets. *Performance Evaluation*, 20:337–357, 1994.
7. G. Ciardo. Discrete-time Markovian stochastic Petri nets. In W. Stewart, editor, *Numerical Solution of Markov Chains*, pages 339–358, Raleigh, NC, 1995. Kluwer.

8. G. Ciardo and A. S. Miner. SMART simulation and Markovian analyzer for reliability and timing. In *Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Tool Descriptions*. Zaragoza, Spain, 1999.
9. J. Couvillion, R. Freire, R. Johnson, Obal W. D. II, M. A. Qureshi, M. Rai, W. H. Sanders, and J. Tvedt. Performability modeling with UltraSAN. *IEEE Software*, 8:69–80, 1991.
10. A. Cumani. ESP – a package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In *Proc. Int. Workshop Timed Petri Nets*, pages 278–184, Turin, Italy, 1985.
11. D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders. Möbius: An extensible framework for performance and dependability modeling. In *Computer Performance Evaluation, Modeling Techniques and Tools*, volume 1786 of *LNCS*, pages 332–336. Springer-Verlag, 2000. 11th Int. Conference, TOOLS 2000.
12. J. Freiheit, A. Zimmermann, R. German, and G. Hommel. Petri net modeling and performability evaluation with TimeNET 3.0. In *Computer Performance Evaluation, Modeling Techniques and Tools*, volume 1786 of *LNCS*, pages 188–202. Springer-Verlag, 2000. 11th Int. Conference, TOOLS 2000.
13. R. Fricks, C. Hirel, S. Wells, C. W. Ro, X. Zhang, and K. S. Trivedi. IDEAS: an integrated design environment for assessment of computer systems and communication networks. Technical report, Duke University, 1998.
14. R. German. Cascaded deterministic and stochastic Petri nets. In *Proc. 3rd Int. Meeting on the Numerical Solution of Markov Chains*, pages 111–130, Zaragoza, Spain, 1999.
15. R. German. Markov regenerative stochastic Petri nets with general execution policies: Supplementary variable analysis and a prototype tool. *Performance Evaluation*, 39:165–188, 2000.
16. R. German. *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley and Sons, 2000.
17. R. German. Iterative analysis of Markov regenerative models. *Journal of Performance Evaluation*, 2001. To appear.
18. R. German, C. Kelling, A. Zimmermann, and G. Hommel. TimeNET: A toolkit for evaluating non-Markovian stochastic Petri nets. *Performance Evaluation*, 24:69–87, 1995.
19. R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. *Performance Evaluation*, 20:317–335, 1994.
20. R. German and M. Telek. Formal relation of Markov renewal theory and supplementary variables in the analysis of stochastic Petri nets. In *Proc. 8th Int. Workshop on Petri Nets and Performance Models*, pages 64–73, Zaragoza, Spain, 1999.
21. S. Haddad, P. Moreaux, and G. Chiola. Efficient handling of phase-type distributions in generalized stochastic Petri nets. In *Application and Theory of Petri Nets 1997, Proc. 18th Int. Conf.*, pages 175–194. Springer-Verlag, 1997.
22. B. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley, 1998.
23. B. R. Haverkort. Markovian models for performance and dependability evaluation. This volume.
24. B. R. Haverkort. Matrix-geometric solution of infinite stochastic Petri nets. In *Proc. IEEE Int. Performance and Dependability Symp.*, pages 72–81, Erlangen, Germany, 1995.
25. A. Heindl and R. German. Performance modeling of IEEE 802.11 wireless LANs with stochastic Petri nets. *Journal of Performance Evaluation*, 2001. To appear.

26. A. Horvath, A. Puliafito, M. Scarpa, and M. Telek. Analysis and evaluation of non-Markovian stochastic Petri nets. In *Computer Performance Evaluation, Modeling Techniques and Tools*, volume 1786 of *LNCS*, pages 171–187. Springer-Verlag, 2000. 11th Int. Conference, TOOLS 2000.
27. IEEE. *802.11: IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, November 1997.
28. R. Jones. Analysis of phase-type stochastic Petri nets with discrete and continuous timing. Technical Report NASA/CR-2000-210296, NASA Langley Research Center, Hampton, VA, USA, 2000.
29. L. Kleinrock. *Queueing Systems*, volume I. John Wiley, 1975.
30. C. Lindemann. DSPNexpress: A software package for the efficient solution of deterministic and stochastic Petri nets. *Performance Evaluation*, 22:3–21, 1995.
31. C. Lindemann. *Performance Modeling with Deterministic and Stochastic Petri Net*. John Wiley, 1998.
32. C. Lindemann and G. Shedler. Numerical analysis of deterministic and stochastic Petri nets with concurrent deterministic transitions. *Performance Evaluation*, 23:565–582, 1996.
33. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley, 1995.
34. M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *Advances in Petri Nets 1986*, LNCS 266, pages 132–145. Springer-Verlag, 1987.
35. M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. John Hopkins University Press, 1981.
36. M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, New York, 1989.
37. A. Puliafito, M. Scarpa, and K. S. Trivedi. Petri nets with k simultaneously enabled generally distributed timed transitions. *Performance Evaluation*, 32(1):1–34, 1998.
38. R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic, 1996.
39. W. H. Sanders, Obal W. D. II, M. A. Qureshi, and F. K. Widjanarko. The UltraSAN modeling environment. *Performance Evaluation*, 24:89–115, 1995.
40. M. Scarpa and A. Bobbio. Kronecker representation of stochastic Petri nets with discrete and continuous PH distributions. In *Proc. 3rd IEEE Int. Performance and Dependability Symp.*, Durham, NC, 1998.
41. M. Telek and A. Bobbio. Markov regenerative stochastic Petri nets with age type general transitions. In *Application and Theory of Petri Nets, Proc. 16th Int. Conf.*, LNCS 935, pages 471–489. Springer-Verlag, 1995.
42. M. Telek, A. Bobbio, L. Jereb, A. Puliafito, and K.S. Trivedi. Steady state analysis of Markov regenerative SPN with age memory policy. In *Quantitative Evaluation of Computing and Communication Systems*, LNCS 977, pages 165–179. Springer-Verlag, 1995.
43. M. Telek, A. Bobbio, and A. Puliafito. Steady state solution of MRSPNs with mixed preemption policies. In *Proc. 2nd IEEE Int. Performance and Dependability Symp.*, pages 106–115, Urbana-Champaign, IL, 1996.
44. M. Telek and A. Horvath. Supplementary variable approach applied to the transient analysis of age-MRSPNs. In *Proc. 3rd IEEE Int. Performance and Dependability Symp.*, pages 44–51, Durham, NC, 1998.

Process Algebra and Markov Chains

Ed Brinksma and Holger Hermanns

Formal Methods and Tools Group, Faculty of Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract. This paper surveys and relates the basic concepts of process algebra and the modelling of continuous time Markov chains. It provides basic introductions to both fields, where we also study the Markov chains from an algebraic perspective, viz. that of Markov chain algebra. We then proceed to study the interrelation of reactive processes and Markov chains in this setting, and introduce the algebra of Interactive Markov Chains as an orthogonal extension of both process and Markov chain algebra. We conclude with comparing this approach to related (Markovian) stochastic process algebras by analysing the algebraic principles that they support.

1 Introduction

The construction of models for the performance and reliability analysis of systems is a difficult task that requires intelligence and experience. Due to the ever increasing size and complexity of systems, such as e.g. embedded and distributed systems, there is a growing need for powerful methods to master the related complications of the modelling task. Performance models do not only become very large, but because of the intricate interplay between (many) system components they can also have a highly irregular structure that is very hard to understand and control. Traditional performance models like Markov chains and queueing networks are widely accepted as simple but effective models in different areas, yet they lack the notion of hierarchical system (de)composition that has proved so useful for conquering the complexity of systems in the domain of functional system properties. The absence of such techniques seriously hampers the adequate modelling of complicated modern systems.

A prominent example of a (class of) formalism(s) for the compositional, hierarchical description and analysis of functional system behaviour is *process algebra* [37,3,28]. It offers a mathematically well-elaborated framework for reasoning about the structure and behaviour of reactive and distributed systems in a compositional way, including abstraction mechanisms that allow for the treatment of system components as black boxes, encapsulating their internal structure. Process algebras are typically equipped with a formally defined structured operational semantics (SOS [51]) that maps process algebra terms onto *labelled transition systems* in a compositional manner. Such labelled transition systems consist of a set of states and a transition relation that describes how the system evolves from one state to another. These transitions are labelled with action

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 183–231, 2001.

names that represent the (inter)actions that may cause the transitions to occur. Such transition systems can be represented as directed edge-labelled graphs, with the states as nodes of and the transitions as edges (labelled with action names).

The labelled transition model is very close to the usual representation of Markov chains as transition systems or automata. Also there system states are connected by directed transition arcs that are labelled. In the case of discrete time Markov chains the labels are probabilities, and in the case of continuous time Markov chains, which are the topic of this paper, the labels are the *rates* that correspond to the exponential distributions that represent the stochastic delays associated with the state transitions. This structural correspondence between the two models motivated the beginning of research in the early 1990's on *stochastic process algebras* [4,27,16], which sought to integrate performance modelling with Markov chains with functional analysis, and to transfer the process algebraic notions of (de)composition and hierarchy to Markov chain theory.

The fruitfulness of this approach to the specification and generation of Markov chains has been demonstrated by a number results. In the stochastic setting, *bisimulation* equivalence [40], a central notion of equivalence for comparing labelled transition systems, has been shown to coincide with *lumpability*, a key concept for the aggregation of Markov chains [27]. Moreover, as bisimulation can be shown to be preserved under system composition operators (algebraically: bisimulation is a *congruence*), Markov chain aggregation can be carried out compositionally, i.e. component-wise. Several (small to medium size) case studies have shown the practicality of this compositional approach, and important progress has been made in exploiting the syntactic structure of specifications for performance analysis purposes, see [26].

In this paper we aim (i) to give an introduction to the essentials of process algebra that are needed for compositional performance modelling, (ii) to introduce the process algebraic approach to Markovian performance modelling using Interactive Markov Chain (IMC) algebra, and (iii) to survey the main algebraic principles that underly related (Markovian) stochastic process algebras. The paper is organised as follows. Section 2 introduces the concepts and features of process algebras, while Section 3 introduces continuous time Markov chains from an algebraic perspective. The algebra of interactive Markov chains is discussed in Section 4. At the end, section 5 compares IMC and other existing stochastic process algebra in terms of the algebraic principles that they support. Section 6, finally, presents the conclusions of the paper.

2 Process Algebra

In this section we introduce a simple process algebraic framework that we will use throughout our paper. Its purpose is to give an intuitive understanding of the key ingredients of process algebra, and prepare for their use in the rest of the paper. We start with the introduction of labelled transition systems, which constitute a simple but powerful operational model for reactive behaviour. We show how these transition systems can be constructed with the aid of three basic operators,

viz. *action-prefixing*, *choice*, and *recursive specification*. Together, these operations give rise to a basic process algebra that can be used for the description of sequential processes. We then extend this algebraic language to concurrent processes with the aid of an operator for *parallel composition*, in combination with an *abstraction operator* to control the scope of the interactions between concurrent processes. It turns out that for many purposes the labelled transition system model is too fine, i.e. there are many different transition systems that display intuitively identical behaviour. This leads us to the definition of useful behavioural equivalences over reactive behaviour, viz. the notions of *strong* and *weak bisimulation*. Finally, we turn to an axiomatic presentation of process algebra by discussing the axiom systems that are induced by the bisimulation equivalences. For a fuller account of the material covered by this section, we refer the reader to the extensive literature of process algebra, e.g. [3,40,5,10].

2.1 Labelled Transition Systems

State-transition diagrams, automata and similar models are widely used to describe the dynamic behaviour of systems. They consists of a set of states S together with a representation of possible state changes. The latter is usually given in the form of some relation (or function) over states, i.e. a subset of the Cartesian product $S \times S$. Intuitively, a pair of states (P, Q) is in this relation if it is possible to change from state P to Q in a single step. Such transition relation are often denoted with an arrow (e.g. $\longrightarrow$), so that $(P, Q) \in \longrightarrow$ can then be conveniently rewritten, in infix notation, as $P \longrightarrow Q$, thus nicely representing the possible state change between P and Q.

In the context of process algebras, transition systems appear in a specific form, viz. that of labelled transition systems. Labelled transition systems form a particular class where state changes are conditioned on occurrences of actions drawn from an *action set*, or *alphabet*, $\mathcal{A}$. A state change between P and Q here entails the occurrence of a related action. Therefore, the transition relation $\longrightarrow$ is a subset of $S \times \mathcal{A} \times S$ rather than a binary relation over just S. Again, it will be convenient to denote $(P, \mathsf{a}, Q) \in \longrightarrow$, using a kind of mixfix notation, as $P \xrightarrow{\mathsf{a}} Q$. Here the action appears as the label of the transition, whence the term 'labelled' transition system.

Definition 1. *A labelled transition system is a triple* $(S, \mathcal{A}, \longrightarrow)$*, where*

- S *is a nonempty set of states,*
- $\mathcal{A}$ *is a set of actions, and*
- $\longrightarrow \subset S \times \mathcal{A} \times S$ *is a set of action labelled transitions.*

In order to use labelled transition systems as an operational model of systems it is common practice to identify a specific initial state P in the transition system where operation starts. A transition system with an initial state is called a process.

Definition 2. *A process is a quadruple* $(S, \mathcal{A}, \longrightarrow, P)$*, where* $(S, \mathcal{A}, \longrightarrow)$ *is a labelled transition system and* $P \in S$ *is the initial state.*

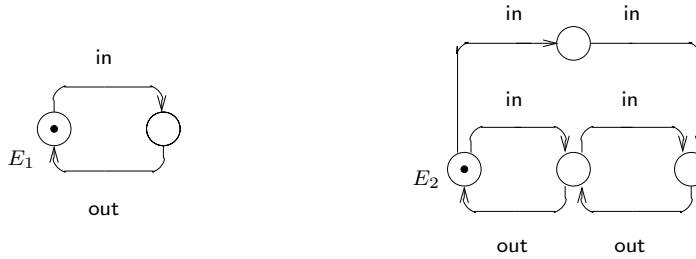


Fig. 1. Two processes.

Example 1. Figure 1 contains two examples of processes. In principle, states are represented as circles labelled with identifiers from S. We adopt the convention, however, to use state labels only if they are required for understanding. We use $\odot$ to denote the initial state. The first process, with initial state E_1, is a simple one-place buffer. It accepts data via the action in and releases them with the action out. The right process is able to buffer two values, but with a slightly unusual restriction. From its initial state E_2 there is a choice between two transitions that are both labelled with in. In the standard interpretation of process algebra this represents a *nondeterministic choice* that is made as part of the execution of the action in, i.e. the receipt of a first datum. The lower branch leads to the usual behaviour of a two-place buffer, whereas after the upper branch no datum can be released, i.e. no out can be executed, before two data have been accepted.

2.2 Basic Processes

Process algebra is a means to specify processes and to reason about them. To achieve this we use an *algebraic language*, based on *combinators*, i.e. operators that compose processes into new ones. The terms of the algebraic language, the *behaviour expressions*, are interpreted as labelled transition systems with a distinguished initial state, i.e. as processes in the sense of Definition 2. This is done using so-called structural operational semantic rules. This semantic interpretation induces equalities between different behaviour expression, yielding an *equational calculus* for reasoning about processes.

We introduce the language by a simple BNF-style grammar. We assume as given a countable set $\mathcal{V}$ of *variables* that are used to express repetitive behaviour, and, as before, a set of actions $\mathcal{A}$. We use a, b, ... for elements of $\mathcal{A}_\tau$. We also assume a distinguished element τ, representing *internal* (or *silent*, or *hidden*) actions, and let $\mathcal{A}_\tau$ denote the set $\mathcal{A} \cup \{\tau\}$.

Definition 3. *Let* $\mathsf{a} \in \mathcal{A}_\tau$ *and* $X \in \mathcal{V}$. *We define the language* PA *as the set of expressions given by the following grammar.*

$$\mathcal{E} ::= \quad 0 \quad | \quad \mathsf{a}.\mathcal{E} \quad | \quad \mathcal{E} + \mathcal{E} \quad | \quad X \quad | \quad [X := \mathcal{E}]_i$$

$[X := \mathcal{E}]$ is a shorthand notation for an arbitrary (finite) set of defining equations of the form $[X_1 := \mathcal{E}_1\ ,\ X_2 := \mathcal{E}_2\ ,\ \ldots\ ,\ X_n := \mathcal{E}_n]$, or in vector notation,

$$\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} := \begin{bmatrix} \mathcal{E}_1 \\ \mathcal{E}_2 \\ \vdots \\ \mathcal{E}_n \end{bmatrix}$$

with $X_i \in \mathcal{V}$, and $\mathcal{E}_i$ complying to the above grammar.

We use E, E_1, E_2, F, ... to range over arbitrary expressions of PA. The intuitive meaning of the language constructs is as follows.

- The terminal symbol 0 describes a *terminated* behaviour that cannot engage in any (inter)action.
- The expression $\mathsf{a}.E$ may interact on action a and afterwards behave as expression E. We shall say that E is *action prefixed* by a.
- The expression $E + F$ combines two alternatives. It either exhibits the behaviour of expression E or the behaviour of expression F. The terminal symbol $+$ is called the *choice operator*. The choice between E and F is resolved in interaction with other processes on the initial actions of E and F.
- The expression $[X := E]_i$ defines a behaviour in terms of the set $[X := E]$ of mutually recursive behaviour definitions . Its meaning is as follows: $[X := E]_i$ behaves as E_i, where the behaviour of the recursion variables is obtained by 'unrolling': wherever a behaviour X_j is reached, it is replaced by (the behaviour of) its definition $[X := E]_j$.

In the sequel, we restrict ourselves to expressions, where each occuring variable X_j is bound by a defining equation $X_j := \ldots$. Such expressions are called *closed expressions*. An expression $E \in \mathsf{PA}$ is closed, if each variable $X_j \in \mathcal{V}$ appearing in E only appears inside the scope of a guarding defining equation set, i.e. inside an expression $[\ldots\ ,\ X_j := \ldots\ ,\ \ldots]_i$. The set of closed expressions is denoted PA^c.

Example 2. An example of an expression that is not closed is $\mathsf{in}.[X_1 := \mathsf{in}.X_2]_1$. The processes of Figure 1 can be specified as follows:

- E_1 is defined by $[X_1 := \mathsf{in.out}.X_1]_1$
- E_2 is defined by $[X_1 := \mathsf{in}.X_2 + \mathsf{in.in.out}.X_2\ ,\ X_2 := \mathsf{in.out}.X_2 + \mathsf{out}.X_1]_1$

We formalise this intuitive interpretation by giving it an operational semantics in terms of labelled transition systems. The style of definition that we use goes back to Plotkin [51], and is usually referred to as structured operational semantics (SOS), since it defines the operational interpretation of a behaviour expression inductively over its syntactical structure.

We define a semantics for closed expressions of PA by mapping the complete language PA^c onto a universal transition system. The state space of this transition system is the set of all closed expressions according to Definition 3. Since

$$\frac{}{\mathsf{a}.E \xrightarrow{\mathsf{a}} E} \qquad \frac{E \xrightarrow{\mathsf{a}} E'}{E + F \xrightarrow{\mathsf{a}} E'} \qquad \frac{F \xrightarrow{\mathsf{a}} F'}{E + F \xrightarrow{\mathsf{a}} F'} \qquad \frac{E_i\{[X := E]_i / X_i\} \xrightarrow{\mathsf{a}} E'}{[X := E]_i \xrightarrow{\mathsf{a}} E'}$$

Table 1. Operational semantic rules for PA^c.

each $E \in \mathsf{PA}^c$ appears somewhere in this transition system the corresponding semantics is determined by the state space reachable from this expression.

The first SOS-rules that we need are given in Table 1. The rules have the format

$$\frac{B}{C} \quad A,$$

to express that *if A holds, then B implies C*, where A,B and C statements about the existence of labelled state transitions. The notation $E\{F/X\}$ is used to represent the result of a simultaneous substitution of each occurrence of variable X by expression F in expression E.

Definition 4. *The universal transition system $\mathcal{U}$ is given by the triple $(\mathsf{PA}^c, \mathcal{A}, \longrightarrow)$, where $\longrightarrow \subset \mathsf{PA}^c \times \mathcal{A} \times \mathsf{PA}^c$ is the least relation satisfying the operational rules in Table 1.*

This definition provides a semantics for each element of $E \in \mathsf{PA}^c$, via the fragment of $\longrightarrow$ reachable from the state E in $\mathcal{U}$. For a closed expression E, we let $Reach(E)$ denote the set of states reachable from E in the universal transition relation $\mathcal{U}$: $Reach(E) = \{E' \mid (E, E') \in T^*\}$ where T is the unlabelled transition relation in $\mathcal{U}$, i.e. $T = \{(F, F') \in \mathsf{PA}^c \times \mathsf{PA}^c \mid \exists a \in \mathcal{A}.\ F \xrightarrow{\mathsf{a}} F'\}$.

Definition 5. *The semantics of a closed behaviour expression $E \in \mathsf{PA}^c$ is a process $(S, \mathcal{A}, \longrightarrow', E)$, where $S = Reach(E)$ and $\longrightarrow' = \longrightarrow \cap (S \times \mathcal{A} \times S)$.*

Because of this definition we can adopt the fairly general convention of identifying a process with its initial state. Closed expressions are thus also called processes.

Example 3. In order to prove that the process E_1 of Figure 1 possesses an outgoing transition labelled with action in we apply the operational rules of Table 1 to construct the following derivation.

$$\frac{\mathsf{in}.\mathsf{out}.[X_1 := \mathsf{in}.\mathsf{out}.X_1]_1 \xrightarrow{\mathsf{in}} \mathsf{out}.[X_1 := \mathsf{in}.\mathsf{out}.X_1]_1}{[X_1 := \mathsf{in}.\mathsf{out}.X_1]_1 \xrightarrow{\mathsf{in}} \mathsf{out}.[X_1 := \mathsf{in}.\mathsf{out}.X_1]_1}$$

2.3 Concurrency and Abstraction

Although basic process algebra suffices, at least in principle, for the description of processes, it is too limited to be of great practical value. When specifying and analysing reactive systems it will often be necessary to conceive of them as the concurrent composition of a number of subprocesses. This can be the case because parallelism is a natural feature of the given system, and we wish to represent it. Or it may be that the properties of a system can be understood better if its behaviour is decomposed into a number of smaller components. Many realistic systems are so complicated, in fact, that they can only be understood in terms of a concurrent composition of components.

A key ingredient of concurrency is the possibility for component processes to interact. Processes interact to achieve a common goal, which means that they somehow have to synchronise their activities, e.g. by exchanging data. Different forms of process interaction have been studied in the literature of process algebra [37,40,10,29]. Distinctive features are asynchronous vs. synchronous and binary vs. multiparty interaction. For our purposes it will be convenient to use the synchronous multiparty interaction as defined, for instance, in the ISO specification language LOTOS [30,5].

We introduce a binary parallel composition operator that is indexed with the set of actions that its component processes have to synchronise on. All other actions, i.e. those that are not in the index set of the composition operator, can be performed independently of the other component process. The basic form of interaction therefore is synchronisation on actions: the execution of a synchronised action is a joint activity of all synchronising processes.

If P and Q are two processes, such synchronous parallel composition is denoted

$$P \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q$$

By varying the set of synchronising actions, parallel composition ranges from *full synchronisation*, when the set comprises all the possible actions, to *arbitrary interleaving*, when the set is the empty (in this case we use the concise notation $P \mid\mid\mid Q$). The intuition behind this operator is summarised by the following informal properties:

- A state change of $P \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q$ is possible if P may change to, say P', on the occurrence of an action a that is not contained in $\{\mathsf{a}_1 \ldots \mathsf{a}_n\}$. The result of the state change is $P' \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q$, since only P has changed state.
- Symmetrically, a state change of $P \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q$ is also possible if Q may change to some Q', on the occurrence of an action a that is not contained in $\{\mathsf{a}_1 \ldots \mathsf{a}_n\}$, resulting in $P \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q'$.
- In order to be able to interact on an action a contained in $\{\mathsf{a}_1 \ldots \mathsf{a}_n\}$, both P and Q have to be able to perform a and thereby evolve to some P' and Q'. If this condition is met $P \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q$ may in a single step change state to $P' \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q'$.
- No other transitions are possible for $P \shortparallel \mathsf{a}_1 \ldots \mathsf{a}_n \shortparallel Q$.

$$\frac{P \xrightarrow{\mathsf{a}} P'}{P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q \xrightarrow{\mathsf{a}} P' \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q} \mathsf{a} \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$$

$$\frac{Q \xrightarrow{\mathsf{a}} Q'}{P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q \xrightarrow{\mathsf{a}} P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q'} \mathsf{a} \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$$

$$\frac{P \xrightarrow{\mathsf{a}} P' \qquad Q \xrightarrow{\mathsf{a}} Q'}{P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q \xrightarrow{\mathsf{a}} P' \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q'} \mathsf{a} \in \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$$

Table 2. Structural operational rules for parallel composition.

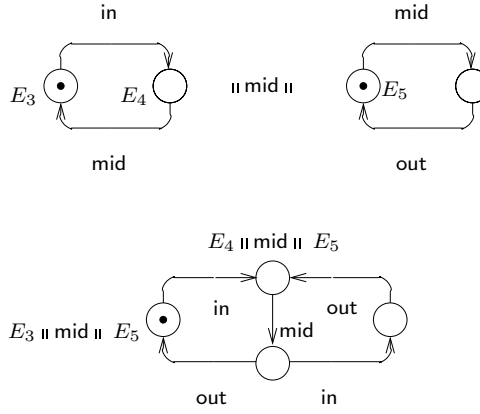


Fig. 2. Parallel composition of two processes.

We extend PA^c with the new operator by stipulating that if P and Q are in PA^c then $P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q$ is in PA^c as well. We can now formalise the above requirements as SOS-rules for the process $P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q$. The first three requirements are reflected in the three derivation rules of Table 2. The last property is automatically fulfilled as the transition relation itself is defined as the *least* relation satisfying the definition, i.e. it does not possess non-derivable transitions.

In the third SOS-rule of Table 2 it can be seen that the result of synchronisation on an action a is a transition of the composite behaviour again labelled with the same action a. This choice (borrowed from [46,5]) is an essential ingredient to enable so-called *multiway* synchronisation, where further processes may synchronise with the a-labelled transition of the composition. This approach, although fairly straightforward, is one of a number of alternatives to interaction and synchronisation, e.g. see [39] for a discussion of this topic.

$$\frac{P \xrightarrow{\mathsf{a}} P'}{\texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P \xrightarrow{\mathsf{a}} \texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P'} \quad \mathsf{a} \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$$

$$\frac{P \xrightarrow{\mathsf{a}} P'}{\texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P \xrightarrow{\tau} \texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P'} \quad \mathsf{a} \in \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$$

Table 3. Structural operational rules for abstraction.

Example 4. Figure 2 shows parallel composition of two processes E_3 and E_5. Below these processes the resulting process $E_3 \parallel \mathsf{mid} \parallel E_5$, obtained by applying the rules of Table 2, is also depicted.

The concept of multiway synchronisation has proven convenient from a specification engineering point of view. It allows for a constraint-oriented style of system specification, where processes add conditions on the occurrence of interactions incrementally using concurrent composition, see e.g. [58]. However, with the operators introduced so far, all actions that occur anywhere in a system specification remain available for further synchronisation with new processes. This is undesirable, since most system design methods try to work with components as black boxes, i.e. as functionality without internal structure. Such an approach calls for mechanisms to *abstract* from internal aspects that are irrelevant at higher design levels.

In process algebra the concept of abstraction must be dealt with in terms of an operator, the *abstraction operator*. The key to this operator is a distinguished action, usually named τ, that symbolises *internal* or *hidden* action, e.g. a state change that does not depend on synchronisation with the environment. Actions other than τ are called *external* or *observable*. For a given process P and actions $\mathsf{a}_1 \ldots \mathsf{a}_n$ abstraction or *hiding* of those actions simply renames them into the internal action τ. We use

$$\texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P$$

to denote this operation. Again, we extend PA^c with a closure condition, viz. that if P is in PA^c then so is $\texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P$. The semantics of the abstraction operator are given by the operational rules in Table 3.

We would like to point out that in a concurrent composition internal actions of one component process should be completely independent of those of the other. Hence, synchronisation on internal actions is ruled out, i.e. τ cannot occur in the index set $\{\mathsf{a}_1 \ldots \mathsf{a}_n\}$ of a composition $P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q$.

Example 5. In Figure 3 we have depicted the result of internalising the action mid in the process $E_3 \parallel \mathsf{mid} \parallel E_5$ by means of abstraction. The behaviour of the resulting process behaves as a two place buffer composed out of two one-place

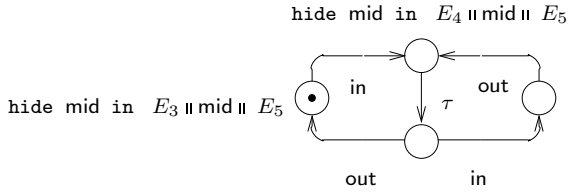


Fig. 3. Abstraction applied to composed processes.

buffers, E_3 and E_5. The first accepts data with action in. These are then passed *internally* to process E_5, which, in turn, can output them. E_3 may accept a second input, but before it can pass it on to E_5 this process has to output the input it accepted earlier.

2.4 Equivalence

An essential part of the development of process algebraic theory has been devoted to the study of suitable notions of behavioural equivalence. Such equivalences are induced by various notions of what constitutes process behaviour, different processes being equivalent iff they display *identical* behaviour. This behaviour-oriented (as opposed to state-oriented) point of view implies that the identity of states cannot be relevant for distinguishing between processes, whereas the labelling of transitions is. All common process algebraic equivalences share this characteristic. Still, there exists an overwhelmingly rich collection of such equivalences. Their variety is caused by the different intuitions about process behaviour. R.J. van Glabbeek has extensively studied the different behavioural equivalences [56, 55]. They are classified according to the observational powers that an observer or experimenter must have to distinguish between different processes. In this paper we will confine ourselves to a particular, but very important class of equivalences, the *bisimulation relations*. We will argue why this is a good class of equivalences for our purposes, in this section and also later on, when probabilities and probability distributions come into play. We start by considering the so-called 'strong' equivalence, where internal and external actions are treated on an equal footing. After that we proceed with the 'weak' equivalence, which abstracts from internal transitions.

Strong equivalence. A labelled transition system can be seen as being essentially an automaton, with its finiteness conditions removed and with only success states. Therefore, the notion language equivalence of automata would seem a natural candidate to be considered for the characterisation of process equivalence. Two transition systems are language equivalent iff they accept the same language, i.e. their (finite) execution traces determine the same set of finite sequences over *Act*. In the context of process algebra this relation is called *trace equivalence*.

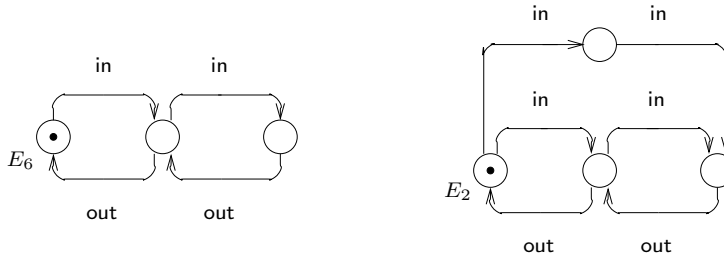


Fig. 4. Two processes with equivalent traces.

Notation. We use $P \xrightarrow{\mathsf{a}_1}\xrightarrow{\mathsf{a}_2} \cdots \xrightarrow{\mathsf{a}_n} P'$ to denote that there exist processes $P_1, P_2, \ldots, P_{n-1}$ such that $P \xrightarrow{\mathsf{a}_1} P_1 \xrightarrow{\mathsf{a}_2} P_2 \cdots P_{n-1} \xrightarrow{\mathsf{a}_n} P'$.

Definition 6. *Let P and Q be processes. P and Q are* strong trace equivalent*, written $P \sim_{tr} Q$, if for all P' and Q',*

$$P \xrightarrow{\mathsf{a}_1}\xrightarrow{\mathsf{a}_2} \cdots \xrightarrow{\mathsf{a}_n} P' \quad \textit{if and only if} \quad Q \xrightarrow{\mathsf{a}_1}\xrightarrow{\mathsf{a}_2} \cdots \xrightarrow{\mathsf{a}_n} Q'.$$

Example 6. Consider the process depicted in Figure 4. E_6 describes the usual two place buffer. E_2 already appeared in Figure 1. Depending on the in branch taken E_2 may loose the possibility to output the first input before a second is accepted. However, both processes are trace equivalent according to Definition 6.

This example gives some insight into the weaknesses of trace equivalence. The process E_2 is not always able to release an input after it has accepted one whereas E_6 always is. This is problematic if E_2 is put in a context where an output is required for synchronisation after every input, (with E_1 of Figure 1, for instance, synchronising on action out and in). E_6 would be able to interact on action out after action in has occurred, whereas E_2 may not, thus forcing a deadlock. In other words, trace equivalence does not preserve deadlocks.

The main reason why trace equivalence is suitable for automata theory, while it does not fit with the process algebraic theory of processes, is the difference between their models of interaction. Automata theory assumes complete control of the automaton over its transitions. In the process algebraic view of processes all observable actions are under the *joint* control of the process and its environment. In this context automata can be seen as processes with only internal actions (but not necessarily labelled with τ), or alternatively, as a process with a completely cooperative environment, i.e. one that is always capable of synchronising on the action of the automatons' choice. The standard interaction between process algebraic processes, however, assumes an interactive resolution of choices, at least between observable transitions. This means that a process cannot select a transition labelled with an observable action if this action is not

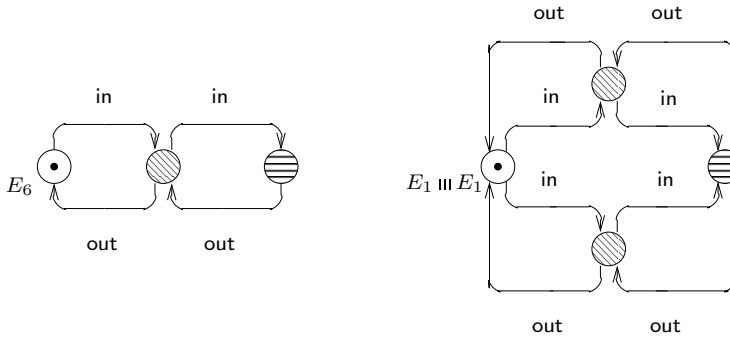


Fig. 5. Two strongly bisimilar processes.

also enabled by the environment. If several such jointly enabled transitions exist, then the choice is made nondeterministically.

In the presence of concurrent composition it is natural that two transition systems should be equivalent if and only if they interact in the same way with arbitrary environments. In view of the above, that means the way in which they constrain the choices between different actions is relevant. This is also referred to as the *branching (time) structure* of processes, as opposed to the *linear (time) structure* of classical automata.

Milner and Park [40,47] have introduced the most important class of equivalence relations that respect the branching structure of a process and therefore are deadlock preserving. This is the class of *bisimulation equivalences* or *bisimilarities*. Two processes are bisimilar if they can simulate each other's behaviour step-by-step. This leads to an inductive definition of bisimulation, based on single steps, that is simple but quite powerful.

Definition 7. *A binary relation $\mathcal{B}$ on PA^c is a strong bisimulation if $(P,Q) \in \mathcal{B}$ implies for all $\mathsf{a} \in \mathcal{A}_\tau$:*

- $P \xrightarrow{\mathsf{a}} P'$ *implies* $Q \xrightarrow{\mathsf{a}} Q'$ *for some Q' such that* $(P',Q') \in \mathcal{B}$,
- $Q \xrightarrow{\mathsf{a}} Q'$ *implies* $P \xrightarrow{\mathsf{a}} P'$ *for some P' such that* $(P',Q') \in \mathcal{B}$.

Two processes, P and Q, are strongly bisimilar, written $P \sim Q$, if they are contained in some strong bisimulation $\mathcal{B}$,i.e. $(P,Q) \in \mathcal{B}$.

Strong bisimilarity, therefore, is the union of all strong bisimulations, i.e.

$$\sim = \bigcup \{\mathcal{B} \mid \mathcal{B} \text{ is a strong bisimulation}\}$$

In words, the above definition says that two states in a transition system are bisimilar if each has transitions labelled with the same actions as the other such that the states after corresponding transitions are bisimilar again.

Example 7. Following Definition 7, the two processes E_6 and $E_1 \,|||\, E_1$ depicted in Figure 5 are bisimilar. To facilitate the inspection of this claim we have shaded

strongly bisimilar states with the same pattern. Note that the right process is obtained by composing in parallel two (one-place) buffers E_1 without any synchronisation. They behave like a two-place buffer E_6 (in fact, they implement a two-place *bag*, but because the data have no identity this is indistinguishable from a buffer).

Strong bisimilarity gives us appropriate means to compare processes with respect to their branching structure. Besides its intuitive content, it also has the correct formal properties that allow for a smooth mathematical treatment.

Proposition 1. *Strong bisimilarity is*

- *an equivalence relation on* PA^c.
- *a strong bisimulation on* PA^c.
- *the* largest *strong bisimulation on* PA^c.

The style of the definition of bisimulation is sometimes called *coinductive*, since it borrows the concept of coinduction from category theory [31]. Roughly speaking, a coinductive definition characterises the largest set satisfying an inductive definition, whereas *in*duction characterises the smallest such set.

Later, we will later also rely on an alternative characterisation of strong bisimilarity, borrowed from [57], which defines $\sim$ as the union of equivalence relations. It makes use of a (boolean) function $\gamma_{\mathrm{o}} : S \times \mathcal{A}_\tau \times 2^S \mapsto \{\mathtt{true}, \mathtt{false}\}$. $\gamma_{\mathrm{o}}(P, \mathsf{a}, C)$ is true iff P can evolve to a state contained in a set of states C by interaction on a.

Definition 8.

$$\gamma_{\mathrm{o}}(P, \mathsf{a}, C) := \begin{cases} \mathtt{true} & \textit{if there is } P' \in C \textit{ such that } P \xrightarrow{\mathsf{a}} P', \\ \mathtt{false} & \textit{otherwise.} \end{cases}$$

With this definition, bisimilarity can be expressed as 'having the same possibilities to interact and make a transition into the same class of behaviours' where these classes are, of course, classes of equivalent behaviour.

Lemma 1. *An equivalence relation $\mathcal{E}$ on S is a strong bisimulation iff $(P, Q) \in \mathcal{E}$ implies for all $\mathsf{a} \in \mathcal{A}_\tau$ and all equivalence classes C of $\mathcal{E}$ that*

$$\gamma_{\mathrm{o}}(P, \mathsf{a}, C) \implies \gamma_{\mathrm{o}}(Q, \mathsf{a}, C).$$

Note that $\mathcal{E}$ is presupposed to be an equivalence relation in this definition, and therefore is symmetric by assumption. Thus, we could equally well replace ' $\implies$ ' by ' $\iff$ ' (or '=') in the above Lemma.

Example 8. If we use  to denote the set of states shaded like  in Figure 5, and similar with  and  , then each of these sets is a class of an equivalence relation $\mathcal{E}$ satisfying Lemma 1. In particular, we compute the following values for each of the states in the respective class. All other combinations return `false`.

$\gamma_{\mathrm{o}}($  $, \mathsf{in},$  $) = \mathtt{true}$ $\qquad$ $\gamma_{\mathrm{o}}($  $, \mathsf{out},$  $) = \mathtt{true}$

$\gamma_{\mathrm{o}}($  $, \mathsf{in},$  $) = \mathtt{true}$ $\qquad$ $\gamma_{\mathrm{o}}($  $, \mathsf{out},$  $) = \mathtt{true}$

Let us now turn our attention to another important property of bisimilarity. Since we work in a setting with composition operators, we must investigate whether $\sim$ induces a proper algebraic notion of equality. In particular, this means that equality should be preserved under composition. In the above example, we have seen that E_6 and $E_1 \,|||\, E_1$ are bisimilar. They both describe the behaviour of a buffer with two places. However, it is not yet clear whether we can use either of them in a larger composition context and obtain again equivalent overall behaviours. This is, of course, a highly desirable property, because it allows normal equational reasoning, replacing a subterm by an equivalent one, without affecting the resulting behaviour. What we need, in general, is the *substitutivity* of an equivalence relation. In algebraic terms this means that we have to show that $\sim$ is a *congruence* (relation) with respect to the operators.

Theorem 1. *Strong bisimilarity is a congruence relation with respect to the operators of* PA^c*, i.e.*

$$
\begin{array}{lcll}
P_1 \sim P_2 & \textit{implies} & \mathsf{a}.P_1 \sim \mathsf{a}.P_2 \\
P_1 \sim P_2 & \textit{implies} & P_1 + P_3 \sim P_2 + P_3, \\
P_1 \sim P_2 & \textit{implies} & P_3 + P_1 \sim P_3 + P_2, \\
P_1 \sim P_2 & \textit{implies} & P_1 \,|[\mathsf{a}_1 \ldots \mathsf{a}_n]|\, P_3 \sim P_2 \,|[\mathsf{a}_1 \ldots \mathsf{a}_n]|\, P_3, \\
P_1 \sim P_2 & \textit{implies} & P_3 \,|[\mathsf{a}_1 \ldots \mathsf{a}_n]|\, P_1 \sim P_3 \,|[\mathsf{a}_1 \ldots \mathsf{a}_n]|\, P_2, \\
P_1 \sim P_2 & \textit{implies} & \mathtt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \mathtt{in}\ P_1 \sim \mathtt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \mathtt{in}\ P_2\,.
\end{array}
$$

Strong bisimilarity shares this substitutivity property with other equivalences, such as trace equivalence. On top of that, it respects the branching structure of processes and therefore preserves deadlocks. Furthermore, it can be defined coinductively. These properties are the main reasons why bisimilarity is a central concept in the theory of process algebraic equivalences. It is easy to define, has a simple proof technique, and is mathematically elegant.

Weak equivalences. So far we have only discussed equivalences that treat internal actions exactly the same way as external actions. In particular, internal actions have to be simulated stepwise to establish strong bisimilarity between two processes. This is counterintuitive, because we ultimately mean to characterise the behaviour of processes by means of their black box, i.e. observable, behaviour. But as internal actions are not observable there seems to be no direct need to be able to simulate each internal transition of an equivalent process.

Example 9. We have discussed before that a serial connection of two one-place buffers as in $\mathtt{hide}\ \mathsf{mid}\ \mathtt{in}\ \ E_3 \,|[\mathsf{mid}]|\, E_5$ behaves like a two-place buffer. However, it is not possible to construct a strong bisimulation between this process and E_6 even though E_6 appears as a canonical representation of a two place buffer. The reason is that we have to (bi)simulate internal $\xrightarrow{\tau}$ transitions that E_6 does not possess (Figure 6).

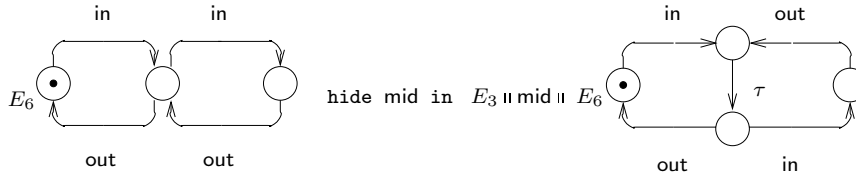


Fig. 6. Not strongly bisimilar processes.

To abstract from internal moves it seems natural to ignore them as far as they do not influence the observable behaviour of a process. To do so, we introduce the notion of an *observable step* of a process, consisting of a single *observable* action preceded and followed by an arbitrary finite number (including zero) of internal steps [40]. This can be seen as deriving a 'weak' transition relation, denoted by $\Longrightarrow$, from the 'strong' transition relation $\longrightarrow$.

Definition 9. *For internal actions,* $\stackrel{\varepsilon}{\Longrightarrow}$ *is defined as the reflexive and transitive closure* $\stackrel{\tau}{\longrightarrow}^*$ *of the relation* $\stackrel{\tau}{\longrightarrow}$*. External weak transitions are then obtained by defining* $\stackrel{\mathsf{a}}{\Longrightarrow}$ *to denote* $\stackrel{\varepsilon}{\Longrightarrow} \stackrel{\mathsf{a}}{\longrightarrow} \stackrel{\varepsilon}{\Longrightarrow}$*.*

Note that a weak internal transition $\stackrel{\varepsilon}{\Longrightarrow}$ is possible without actually performing an internal action, because $\stackrel{\tau}{\longrightarrow}^*$ contains the reflexive closure, i.e. the possibility not to move at all. In contrast, a weak external transition $\stackrel{\mathsf{a}}{\Longrightarrow}$ must contain exactly one transition $\stackrel{\mathsf{a}}{\longrightarrow}$ preceded and followed by arbitrary (possibly empty) sequences of internal moves. We use $\mathcal{A}_\varepsilon$ to range over visible actions and ε, i.e. $\mathcal{A}_\varepsilon = \mathcal{A} \cup \{\varepsilon\}$.

Example 10. For the processes E_6 and $\mathtt{hide\ mid\ in}\ \ E_3 \,|\!|\, \mathsf{mid} \,|\!|\, E_5$ the weak transition relation is represented by the arrows in Figure 7.

With this relation, weak trace equivalence and weak bisimilarity are obtained by simply replacing strong by weak transitions in Definition 6 and Definition 7, respectively. Since weak trace equivalence inherits the problems of its strong counterpart, we are not interested in this relation here.

Definition 10. *A binary relation* $\mathcal{B}$ *on* PA^c *is a weak bisimulation if* $(P, Q) \in \mathcal{B}$ *implies for all* $\mathsf{a} \in \mathcal{A}_\varepsilon$:

- $P \stackrel{\mathsf{a}}{\Longrightarrow} P'$ *implies* $Q \stackrel{\mathsf{a}}{\Longrightarrow} Q'$ *for some* Q' *such that* $(P', Q') \in \mathcal{B}$,
- $Q \stackrel{\mathsf{a}}{\Longrightarrow} Q'$ *implies* $P \stackrel{\mathsf{a}}{\Longrightarrow} P'$ *for some* P' *such that* $(P', Q') \in \mathcal{B}$.

Two processes, P *and* Q*, are weakly bisimilar, written* $P \approx Q$*, if they are contained in some weak bisimulation* $\mathcal{B}$*.*

Weak bisimilarity has the same basic properties as strong bisimilarity (cf. Proposition 1).

Proposition 2. *Weak bisimilarity is*

- *an equivalence relation on* PA^c.
- *a weak bisimulation on* PA^c.
- *the* largest *weak bisimulation on* PA^c.

In addition it is a congruence relation for all operators, except for the choice operator.

Lemma 2. *Weak bisimilarity is a congruence with respect to prefix, parallel composition and abstraction, but not with respect to choice.*

In order to illustrate that $\approx$ is not a congruence with respect to choice we consider the following counterexample [40]. By Definition of $\approx$ it is obvious that

$$\tau.\mathsf{a}.0 \;\approx\; \mathsf{a}.0$$

holds. Supposing that $\approx$ is a congruence with respect to choice, we can conclude that

$$\underbrace{\tau.\mathsf{a}.0 + \mathsf{b}.0}_{P} \;\approx\; \underbrace{\mathsf{a}.0 + \mathsf{b}.0}_{Q}$$

must also hold. But in P there is a transition labelled τ to $\mathsf{a}.0$. In other words, $P \stackrel{\varepsilon}{\Longrightarrow} \mathsf{a}.0$. In order to satisfy Definiton 10, there need to be some Q' with $Q \stackrel{\varepsilon}{\Longrightarrow} Q'$, satisfying $\mathsf{a}.0 \approx Q'$. But this is not the case. The only candidate for Q' – Q itself – obviously differs from $\mathsf{a}.0$. Thus the assumed congruence property turns out to be false.

The problem is that initial internal transitions need to be treated slightly stronger. To heal this problem, we refine weak bisimilarity.

Definition 11. *P and Q are weakly congruent, written $P \approx^c Q$ iff for all $\mathsf{a} \in \mathcal{A}_\tau$*

1. *$P \stackrel{\mathsf{a}}{\longrightarrow} P'$ implies $Q \stackrel{\varepsilon}{\Longrightarrow} \stackrel{\mathsf{a}}{\longrightarrow} \stackrel{\varepsilon}{\Longrightarrow} Q$ for some Q' with $P' \approx Q'$,*
2. *$Q \stackrel{\mathsf{a}}{\longrightarrow} Q'$ implies $P \stackrel{\varepsilon}{\Longrightarrow} \stackrel{\mathsf{a}}{\longrightarrow} \stackrel{\varepsilon}{\Longrightarrow} P'$ for some P' with $P' \approx Q'$.*

Weak congruence and weak bisimilarity only differ in the treatment of initial internal steps of P and Q. Weak bisimilarity requires that an internal transition $\stackrel{\tau}{\longrightarrow}$ is simulated by a weak transition $\stackrel{\varepsilon}{\Longrightarrow}$, which includes the possibility that no internal transition has to be carried out (cf. Definition 9). For initial behaviours, weak congruence strengthens this requirement. It requires that an internal transition $\stackrel{\tau}{\longrightarrow}$ has to be matched by $\stackrel{\tau}{\longrightarrow}^* \stackrel{\tau}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^*$, i.e. by at least on internal transition $\stackrel{\tau}{\longrightarrow}$.

Theorem 2. *Weak congruence is substitutive with respect to the operators of* PA^c*, i.e.*

$$\begin{aligned}
&P_1 \simeq P_2 \quad \textit{implies} && \mathsf{a}.P_1 \simeq \mathsf{a}.P_2\\
&P_1 \simeq P_2 \quad \textit{implies} && P_1 + P_3 \simeq P_2 + P_3,\\
&P_1 \simeq P_2 \quad \textit{implies} && P_3 + P_1 \simeq P_3 + P_2,\\
&P_1 \simeq P_2 \quad \textit{implies} && P_1 \,|\!|\, \mathsf{a}_1 \ldots \mathsf{a}_n \,|\!|\, P_3 \simeq P_2 \,|\!|\, \mathsf{a}_1 \ldots \mathsf{a}_n \,|\!|\, P_3,\\
&P_1 \simeq P_2 \quad \textit{implies} && P_3 \,|\!|\, \mathsf{a}_1 \ldots \mathsf{a}_n \,|\!|\, P_1 \simeq P_3 \,|\!|\, \mathsf{a}_1 \ldots \mathsf{a}_n \,|\!|\, P_2,\\
&P_1 \simeq P_2 \quad \textit{implies} && \mathtt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \mathtt{in}\ P_1 \simeq \mathtt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \mathtt{in}\ P_2\,.
\end{aligned}$$

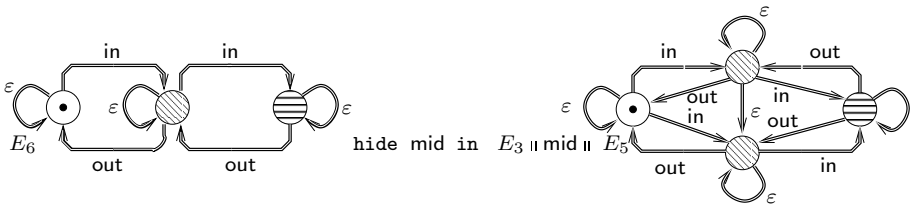


Fig. 7. Two weakly congruent processes.

Indeed, weak congruence is unique in the sense that it turns out to be the *coarsest* congruence contained in weak bisimilarity, as a consequence of the following lemma.

Lemma 3. *$E_1 \simeq E_2$ iff, for each $E_3 \in \mathsf{IMC}^c$, $E_1 + E_3 \approx E_2 + E_3$ and $E_3 + E_1 \approx E_3 + E_2$.*

As a result, we have obtained two substitutive equivalence notions on PA^c: strong bisimilarity and weak congruence, a distinguished subset of weak bisimilarity. The interrelation between these equivalences is expressed in the following lemma.

Lemma 4. $\sim \subset \simeq \subset \approx$.

Example 11. We have pointed out that the processes E_6 and $\texttt{hide mid in}\ \ E_3 \,|[\,]|\, \text{mid} \,|[\,]|\, E_5$ are not strongly bisimilar. But they are weakly bisimilar according to Definition 10. To illustrate this, Figure 7 shows bisimilar states shaded with the same pattern. A crucial aspect is that the weak internal transition $\overset{\varepsilon}{\Longrightarrow}$ of the right process can be simulated by the left process because $\overset{\varepsilon}{\Longrightarrow}$ contains the reflexive closure.

This example shows that weak congruence is an appropriate notion to compare the behaviour of components when internal actions are present. Furthermore, it is indeed a congruence, it is defined coinductively, and it preserves *observable* deadlocks, i.e. the (in)capacity in a state to execute weak transitions labelled by an observable action.

Nevertheless, weak congruence is not as undisputed among the vast number of weak relations as strong bisimilarity is among the strong relations. Some, e.g. van Glabbeek&Weijland [54] and Montanari&Sassone [41] point out that weak bisimilarity is too coarse to preserve the precise branching structure of a process. Others, like Darondeau [14], Valmari [53], de Nicola&Hennessy [44], Parrow&Sjödin [48], Cleaveland&Natarjan [42], as well as Brinksma *et al.* [9] define again coarser equivalences and argue that these relations characterise the observable behaviour of processes better than $\simeq$ does.

It may be worth to point out that it is desirable to have an equivalence notion that is as coarse as possible, given the criteria for equivalent behaviour (in the form of required preservation properties, for example). An equivalence that is too fine will distinguish between too many processes, and therefore satisfy fewer equations, making verification of certain systems more difficult, if not impossible. From this point of view, (fair) testing equivalences seem to be the right choice [10,49,35,44,42,9] – if one is interested in the preservation of observable deadlocks. Essentially, each of the proposed relations is the coarsest in its category, each corresponding to a natural scenario of what an observer is able to test. However, we will not treat them here, basically they do not have coinductive definitions, which we will need for our stochastic extensions later.

2.5 Algebra of Processes

The previous section has illustrated the usefulness of congruences, i.e. equivalences that are substitutive with respect to the language operators. In the presence of such a congruence, it is interesting to investigate in which sense the congruence can be characterised on PA^c by a set of equational laws.

Example 12. An example of an equational law is the commutativity law $E+F = F+E$. The intuitive meaning of this law is as follows: Whenever a pattern of the form $E+F$ can be found in an expression, it can be replaced by $F+E$. E and F play the roles of meta variables and can be instantiated by arbitrary expressions of PA^c. In this way we may transform $\mathsf{b}.(\mathsf{a}.0 + \mathsf{c}.\mathsf{d}.0)$ into $\mathsf{b}.(\mathsf{c}.\mathsf{d}.0 + \mathsf{a}.0)$: We instantiate $E \equiv \mathsf{a}.0$ and $F \equiv \mathsf{c}.\mathsf{d}.0$ and afterwards substitute $F + E$ for $E + F$.

Formally, an equational law is a pair of expressions of PA^c connected with the symbol '=' where either of the expressions, may contain some 'meta variables' such as E, F, and so on. In technical terms, a law (or a set of laws) induces an equivalence on PA^c, or more precise a *congruence*, since we are allowed to replace sub-expressions inside larger expressions, as in the above example.

The question arises in what sense such an induced congruence is related to the congruences we have defined on the semantics of PA^c, i.e. strong bisimilarity and weak congruence. Indeed we are aiming to provide laws that are *sound* with respect to, say, strong bisimilarity. A law is sound with respect to an equivalence, if any application of the law does not alter the equivalence class of the expression. The converse direction is called *completeness*. A set of laws is complete with respect to an equivalence, if two expressions can be transformed into each other by (iterative) application of laws whenever they are equivalent.

Example 13. The law $0 = 0$ is sound for any equivalence relation on PA^c. However, this law is, as it stands alone, far from providing a complete set of laws for any nontrivial equivalence. On the other hand, a law $E = F$ is complete for any equivalence relation on PA^c, but it is sound only for the trivial relation $\mathsf{PA}^c \times \mathsf{PA}^c$ that equates all processes.

(C)	$E + F = F + E$
(A)	$(E + F) + G = E + (F + G)$
(I)	$E + E = E$
(N)	$E + 0 = E$

Table 4. Axioms for strong bisimilarity.

(C)	$E + F = F + E$
(A)	$(E + F) + G = E + (F + G)$
(I)	$E + E = E$
(N)	$E + 0 = E$
$(\tau 1)$	$\mathsf{a}.\tau.E = \mathsf{a}.E$
$(\tau 2)$	$E + \tau.E = \tau.E$
$(\tau 3)$	$\mathsf{a}.(E + \tau.F) + \mathsf{a}.F = \mathsf{a}.(E + \tau.F)$

Table 5. Axioms for weak congruence.

So, our ultimate goal is to provide sets of laws that are *sound as well as complete* with respect to strong bisimilarity, respectively weak congruence. We shall say that such a set *axiomatises* the respective congruence. This is what turns the set PA^c into a true *algebra*.

To give a flavor of this algebraic view on PA^c, Table 4 lists the main equational laws axiomatising strong bisimilarity. The laws state that the choice operator is commutative (C), associative (A), idempotent (I), and that 0 is the neutral element of choice (N). There are four more laws needed to handle recursion and we refer to [38] or [19] for a detailed explanation.

Turning our attention to weak congruence, Table 5 presents a set of laws that form the core of an axiomatisation of weak congruence on PA^c. The upper part of these laws is literally copied from Table 4. This should not be surprising, because strong bisimilarity is a subset of weak congruence (cf. Lemma 4) and therefore every pair that can be proven to be strongly bisimilar has to be weakly congruent, as well. This is a striking reason why the axiomatisation of weak congruence is an extension of the axiomatisation of strong bisimilarity. The law $(\tau 1)$ allows one to skip (action guarded) internal steps. Law $(\tau 2)$ and $(\tau 3)$ expresses that certain behaviours that are preceded by an internal step can happen instantly provided a τ-guarded copy persists.

We shall now discuss the additional operators we have defined on PA^c, abstraction and parallel composition. We present a set of additional laws, that allow one to rewrite parallel composition, as well as abstraction into the basic operators of PA^c. Table 6 lists the necessary laws. Law (X) is usually called the *expansion law*. It states that non-synchronising actions of components can be simply interleaved. Either the left ($\mathsf{a}_j \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$), or the right component

$$(X)\ \sum \mathsf{a}_j.P_j \between \mathsf{a}_1 \ldots \mathsf{a}_n \between \sum \mathsf{b}_l.Q_l = \sum_{\mathsf{a}_j \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}} \mathsf{a}_j.(P_j \between \mathsf{a}_1 \ldots \mathsf{a}_n \between Q) + \sum_{\mathsf{b}_l \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}} \mathsf{b}_l.(P \between \mathsf{a}_1 \ldots \mathsf{a}_n \between Q_l) + \sum_{\mathsf{a}_j = \mathsf{b}_l \in \{\mathsf{a}_1 \ldots \mathsf{a}_n\}} \mathsf{a}_j.(P_j \between \mathsf{a}_1 \ldots \mathsf{a}_n \between Q_l)$$

$$(H1)\quad \texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ 0 = 0$$

$$(H2)\quad \texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ \mathsf{a}.P = \mathsf{a}.\texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ P \quad \textit{provided}\ \mathsf{a} \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$$

$$(H3)\quad \texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ \mathsf{a}.P = \tau.\texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ P \quad \textit{provided}\ \mathsf{a} \in \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$$

$$(H4)\quad \texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ P + Q = \texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ P + \texttt{hide}\ \mathsf{a}_1 \ldots \mathsf{a}_n\ \texttt{in}\ Q$$

Table 6. Axioms for rewriting parallel composition and abstraction.

($\mathsf{b}_l \notin \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$) performs a non-synchronising action. In case of synchronisation ($\mathsf{a}_j = \mathsf{b}_l \in \{\mathsf{a}_1 \ldots \mathsf{a}_n\}$), both partner evolve further.

The laws $(H1) - (H4)$ are very simple. They say that abstraction distributes over termination, over choice and over action prefix, where, according to $(H3)$ action a is internalised if it appears in the set $\{\mathsf{a}_1 \ldots \mathsf{a}_n\}$ of actions. With these laws, parallel composition and abstraction can be shifted arbitrarily deep into a specification, until either 0 or some variable X is reached. This is enough to ensure completeness for a language that includes abstraction and parallel composition (but where the use of recursion is restricted, see e.g. [19]).

This concludes our brief summary how bisimulation can be characterised axiomatically. These axioms are particularly handy to reason about PA^c in an abstract fashion. One can capture the essence of the language just by agreeing (or disagreeing) with a particular set of axioms. It is important to mention that a highly influential strand of process algebra research (also known as the *Dutch school* [1,3]) proceeds the other way round than the way we chose here. This school presupposes a specific equational theory, and then investigates the models and equivalences needed to match this theory. We will follow this way in Section 4 when we introduce an algebra of Interactive Markov Chains.

3 Markov Models

Continuous time Markov chains (MCs) are a particular class of stochastic models that forms a cornerstone of contemporary performance and dependability evaluation methodology [25,18]. This section reviews the main ingredients of MCs from an algebraic perspective, i.e. we proceed similar to the preceding section. After introducing Markov chains and their basic properties, we discuss a bisimulation style equivalence on such chains, which is also known as *lumpability*. We discuss equational properties of this equivalence by developing a small algebra

of MCs, to illustrate the relation to standard process algebra. Broad background material on Markov chains and their analysis can be found in [18].

3.1 Continuous Time Markov Chains

A continuous time Markov chain is a stochastic process $\{X(t) \mid t \in \mathbb{R}\}$ with discrete state space satisfying the so called Markov property. This means that the random variable X takes values of some discrete set S (the state space), and the values of X vary continuously as time passes, satisfying that for $t_n + \Delta t > t_n > t_{n-1} > t_{n-2} > \ldots > t_0$,

$$\begin{aligned} &Prob\{X(t_n + \Delta t) = P' \mid X(t_n) = P, X(t_{n-1}) = P_{t_{n-1}}, \ldots, X(t_0) = P_{t_0}\} \\ &= Prob\{X(t_n + \Delta t) = P' \mid X(t_n) = P\} \\ &= Prob\{X(\Delta t) = P' \mid X(0) = P\}. \end{aligned}$$

Thus, the fact that the process was in state P_{n-1} at time t_{n-1}, in state P_{n-2} at time t_{n-2}, and so on, up to the fact that it was in state P_0 at time t_0 is completely irrelevant. The state $X(t_n)$ contains all relevant history information to determine the random distribution on S at time t_{n+1}. This probability is independent of the actual time instant t_n (or t' or 0) of observation. Nevertheless it *does* depend on the length of the time *interval* Δt. It requires some limit calculation to deduce that we are facing a *linear dependence* [34]. More precise, for every pair of states P and P', there is some parameter λ such that (for small Δt)

$$Prob\{X(\Delta t) = P' \mid X(0) = P\} = \lambda \Delta t + o(\Delta t)$$

where $o(\Delta t)$ subsumes the probabilities to pass through intermediate states between P and P' during the interval Δt. The quantity λ is thus a transition *rate*, a nonnegative real value that scales how the (one step) transition probability between P and P' increases with time. Here, we have implicitly assumed that state P is different from P'. If, otherwise, state P and P' coincide, the probability to stay in state P during an interval Δt (and hence $Prob\{X(\Delta t) = P \mid X(0) = P\}$) decreases with time, starting from 1 if $\Delta t = 0$. The corresponding transition rate is thus a negative real value. It is implicitly determined by the increasing probability to leave state P; that is, it is the negative sum of the respective transition rates.

The probabilistic behaviour of a MC is completely described by the initial state occupied at time 0 (or an initial probability distribution on S) and the transition rates between distinct states. We therefore fix a MC by means of a specific transition relation, $P \stackrel{\lambda}{\longrightarrow} P'$, defined on a state space S, together with an initial state P.

Definition 12. *A Markov transition system is a tuple $(S, \longrightarrow)$, where S is a nonempty set of states, and $\longrightarrow$ is a* Markov transition relation*, a subset of $S \times \mathbb{R}^+ \times S$. A Markov chain is a triple $(S, \longrightarrow, P)$, where $(S, \longrightarrow)$ is a Markov transition system, and $P \in S$ is the initial state.*

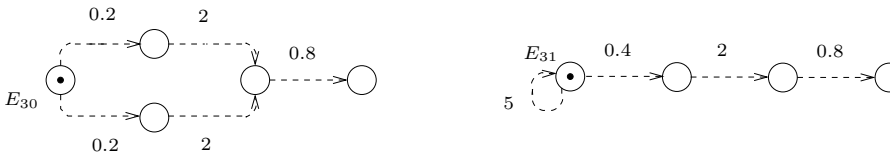


Fig. 8. Two Markov chains

Example 14. Figure 8 contains two examples of Markov chains, E_{30} and E_{31}.

The time of staying in a particular state of S is a random variable, usually called sojourn time. The sojourn time for any state of a MC is known to be *exponentially distributed*. We highlight the following important properties enjoyed by exponential distributions. Let D, D_1, and D_2 denote exponentially distributed random variables.

(A) An exponential distribution $Prob\{D \leq t\} = 1 - e^{-\lambda t}$ is characterised by a single parameter λ, a positive real value, usually referred to as the *rate* of the distribution. The mean duration of this delay amounts to $1/\lambda$ time units.

(B) The class of exponential distribution is the only class of *memoryless* continuous probability distribution. The remaining delay after some time t_0 has elapsed is a random variable with the same distribution as the whole delay:

$$Prob\{D \leq t + t_0 \mid D > t_0\} \;=\; Prob\{D \leq t\}. \tag{1}$$

(C) The class of exponential distributions is closed under minimum, which is exponentially distributed with the sum of the rates:

$$Prob\{\min(D_1, D_2) \leq t\} = 1 - e^{-(\lambda_1+\lambda_2)t} \tag{2}$$

if D_1 (D_2, respectively) is exponentially distributed with rate λ_1 (λ_2).

(D) The probability that D_1 is smaller than D_2 (and vice versa) can be directly derived from the respective rates:

$$Prob\{D_1 < D_2\} = \frac{\lambda_1}{\lambda_1 + \lambda_2} \tag{3}$$

$$Prob\{D_2 < D_1\} = \frac{\lambda_2}{\lambda_1 + \lambda_2}. \tag{4}$$

(E) The continuous nature of exponential distributions ensures that the probability that both delays elapse at the same time instant is zero.

Property (C) explains why the sojourn time for a state s is exponentially distributed. Every transition $s \xrightarrow{\lambda} s'$ leaving state s can be seen to have an exponentially distributed random variable (with parameter λ) associated that

governs when this transition may happen. A race is assumed to exist between several transitions, i.e., they compete for a state change. The sojourn time in s ends as soon as the first transition is ready to occur, inducing a state change. Due to property (C) this sojourn time is exponentially distributed with the sum of the rates of the transitions involved. Property (D) determines the probability of a specific transition to win such a race.

3.2 Equivalence

Strong and weak bisimilarities, as introduced in Section 2, are central in the theory of process algebraic equivalences. Apart from their theoretical importance, a practical merit is the possibility of behaviour preserving state space aggregation. This is achieved by neglecting the identity of states in favour of equivalence classes of states exhibiting identical behaviours. We follow the same spirit in the context of Markov chains.

Strong equivalence. For a given chain, assume that we are only interested in probabilities of equivalence classes of states with respect to some equivalence $\sim$ (that we are aiming to define) instead of probabilities of states. Any such equivalence preserving view on a Markov chain gives rise to an aggregated stochastic process $\widetilde{X} = \{\widetilde{X}(t) | t \in T\}$. It can be defined on the state space $S/\!\sim$, the set of the equivalence classes with respect to $\sim$, by

$$Prob\{\widetilde{X}_t = C\} := Prob\{X(t) \in C\} \qquad \text{for each } C \in S/\!\sim. \tag{5}$$

$\widetilde{X}$ is a discrete state space stochastic process, but it is not necessarily a MC. However sufficient conditions exist such that $\widetilde{X}$ is again a time homogeneous MC. They impose restrictions on the shape of the sets C and are known as lumping conditions, see [34]. We approach them from a different viewpoint, namely by constraints on the equivalence $\sim$, similar to [11,27]. Anticipating the technical details, we achieve that $\widetilde{X}$ is a MC, if $\sim$ is a variant of bisimulation. The difficulty is that we have to equate not only qualities but also *quantities*, for example transition rates of moving from one state to an equivalence class. In contrast, bisimilarity only talks about a (logical) quality: Either there is a move from a state into a class possible or it is impossible, but *tertium non datur*.

The bridge to *quantify* strong bisimilarity is an alternative characterisation of strong bisimilarity that we have mentioned as Lemma 1. To recall its essentials, note that it uses a predicate $\gamma_{\text{O}} : S \times \mathcal{A}_\tau \times 2^S \mapsto \{\texttt{true}, \texttt{false}\}$ that is true iff P can evolve to a state contained in a set of states C (by interaction on action a). Bisimilarity then occurs as the union of all equivalence relations that equate two states if they posses the same γ_{O} values (for each possible combination of action a and equivalence class C).

We follow this style of definition but replace the predicate γ_{O} by a (nonnegative) real-valued function $\gamma_{\text{M}} : S \times 2^S \mapsto \mathbb{R}^+$, that calculates the cumulative rate to reach a set of states C from a single state R:

$$\gamma_{\text{M}}(R, C) = \sum \{\!|\lambda | R \xrightarrow{\lambda} R' \text{ and } R' \in C|\!\}.$$

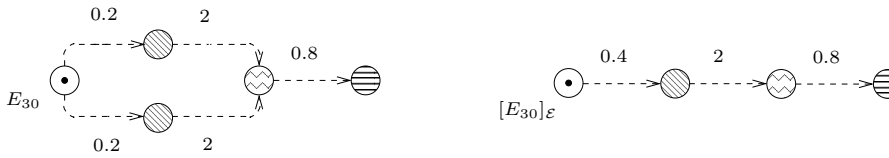


Fig. 9. A Markov chain and its aggregated representative

In this definition we let $\sum \{\!|\ldots|\!\}$ denote the sum of all elements in a multiset (of transition rates), where $\{\!|\ldots|\!\}$ delimits this multiset. The need for this notational burden is best explained by means of an example.

Example 15. Considering Figure 8, the cumulative rate to reach any state in S from state E_{30} is $\gamma_{\mathrm{M}}(E_{30}, S) = \sum \{\!|0.2, 0.2|\!\}$ which amounts to 0.4 due to our definition.

We are now ready to lift bisimilarity to the setting of Markov chains.

Definition 13. *For a given Markov chain $(S, \longrightarrow, P)$, an equivalence relation $\mathcal{E}$ on S is a Markov bisimulation iff $P\mathcal{E}Q$ implies that for all equivalence classes C of $\mathcal{E}$ it holds that*

$$\gamma_{\mathrm{M}}(P, C) \leq \gamma_{\mathrm{M}}(Q, C)$$

Two states P and Q are Markov bisimilar, written $P \sim_{\mathrm{M}} Q$, if (P, Q) is contained in some Markov bisimulation $\mathcal{E}$.

Thus $\sim_{\mathrm{M}}$ is the union of all such equivalences. Indeed, it is itself a Markov bisimulation and therefore the largest such relation (Note that as in Lemma 1 the relation $\mathcal{E}$ is presupposed to be an equivalence, and thus we could write '$=$' instead of '$\leq$').

Definition 14. *For a given Markov chain $(S, \longrightarrow, P)$ and a Markov bisimulation $\mathcal{E}$ on S, define an aggregated chain $(S/\mathcal{E}, \longrightarrow_{\mathcal{E}}, [P]_{\mathcal{E}})$ where the Markov transition relation $\longrightarrow_{\mathcal{E}}$ is given by*

$$[P']_{\mathcal{E}} \xrightarrow{\lambda}_{\mathcal{E}} [Q']_{\mathcal{E}} \qquad \textit{iff} \qquad \gamma_{\mathrm{M}}(P', [Q']_{\mathcal{E}}) = \lambda.$$

Example 16. With the notation introduced in Chapter 2 each of the sets  ,  ,  and  appearing in Figure 9 is a class of an equivalence relation $\mathcal{E}$ on the state space of E_{30} satisfying Definition 13. In particular, we compute the values

$\gamma_{\mathrm{M}}($  $) = 0.4$ $\gamma_{\mathrm{M}}($  $) = 2$ $\gamma_{\mathrm{M}}($  $) = 0.8$

for the states in the respective classes, all other values of γ_{M} are zero. The aggregated Markov chain $[E_{30}]_{\mathcal{E}}$ obtained by applying Definition 14 is depicted on the right.

Theorem 3. *Let P be a Markov chain, describing the CTMC X and let $\mathcal{E}$ be a Markov bisimulation on the state space of P. The aggregated chain $P_{\mathcal{E}}$ describes a homogeneous CTMC $\{\widetilde{X}(t) \mid t \in \mathbb{R}\}$ such that for all equivalence classes C of $\mathcal{E}$,*

$$Prob\{\widetilde{X}(t) = C\} = Prob\{X(t) \in C\}.$$

Proof. The conditions imposed on a Markov bisimulation can be matched with the definition of lumpability [34], see [27].

As a particular consequence, the stochastic process induced by factorising with respect to a Markov bisimulation is again a MC.

As mentioned above this kind of aggregation is known as *lumping*. Lumping is usually formulated with respect to a *suitable* partitioning of the state space. Here, we have defined a *suitability criterion* in a coinductive way. Our partitioning is obtained by means of a factorisation with respect to a bisimulation. This coinductive definition can be exploited for an algorithmic computation of the best possible lumping, see [24].

Weak equivalence. Hitherto we have studied only *strong* bisimilarity on Markov chains. It seems to be equally worthwhile to investigate *weak* bisimilarity. For this purpose, several questions have to be addressed.

First, what is the counterpart of a *weak transition* in terms of Markovian transitions? In the non-stochastic setting we have used a weak transition relation to successfully define weak bisimilarity. It has been based on the distinction between internal actions (labelled τ) and external, observable actions. Such a distinction is not obvious for Markovian chains, because there is no notion of interaction with the external environment.

We may therefore refuse to think about weak relations on Markovian chains at all. Alternatively we may decide that either none, or all of the Markovian transitions are internal. In the former case, a weak Markovian bisimulation will not differ from its strong counterpart, because there is no internal transition that could be abstracted away. So, what about assuming that all Markovian transitions are internal? The corresponding weak transition relation would then combine sequences of Markovian transitions into a single 'weak' transition, in the same way as $\stackrel{\varepsilon}{\Longrightarrow}$ combines sequences of $\stackrel{\tau}{\longrightarrow}$ transitions. For instance, a sequence $P \stackrel{\lambda}{-\!\!\rightarrow} P' \stackrel{\mu}{-\!\!\rightarrow} P''$ could be combined to a weak transition from P to P'' with a parameter ν. This parameter subsumes the exponentially distributed sojourn times in P and P', and it may, in general, be defined as a function $\phi(\lambda, \mu)$.

Unfortunately, the sequence of two (or more) exponentially distributed delays is no longer exponentially distributed. So, any particular choice of a function ϕ will introduce (a possibly severe) error in the model. In other words, replacing a sequence of Markovian transitions by a single *weak* Markovian transitions will lead to a CTMC where it is impossible to reconstruct the stochastic behaviour of the original chain. A result similar to Theorem 3 is thus not possible for any kind of weak Markovian bisimulation. Approximate solutions to this problem have been proposed, and we refer to [27] for a discussion of this topic.

$$\frac{}{(\lambda).E \xrightarrow{\lambda} E} \qquad \frac{E \xrightarrow{\lambda} E'}{E + F \xrightarrow{\lambda} E'} \qquad \frac{F \xrightarrow{\lambda} F'}{E + F \xrightarrow{\lambda} F'} \qquad \frac{E_i\{[X := E]_i / X_i\} \xrightarrow{\lambda} E'}{[X := E]_i \xrightarrow{\lambda} E'}$$

Table 7. Operational semantic rules for MC^c.

3.3 Algebra of Markov Chains

We now develop an algebraic view on Markov chains. To do so, we first define a small, action less algebra, that allows us to generate Markov chains.

Definition 15. *Let $\lambda \in \mathbb{R}^+$ and $X \in \mathcal{V}$. We define the language MC^c as the set of* closed *expressions given by the following grammar.*

$$\mathcal{E} ::= \quad 0 \quad | \quad (\lambda).\mathcal{E} \quad | \quad \mathcal{E} + \mathcal{E} \quad | \quad X \quad | \quad [X := \mathcal{E}]_i$$

The expression $(\lambda).E$, a *delay prefixed* expression, has the intuitive meaning that a process $(\lambda).P$ has to delay for some time before turning into the process P. The amount of time needed to delay is determined by λ, which serves as a parameter of an exponential distribution. In other words, the probability that $(\lambda).P$ has to wait less than t units of time before turning into process P equals $1 - e^{\lambda t}$.

The semantics of MC^c is depicted in Table 7, defining a Markov transition relation $\longrightarrow \subset \mathsf{MC}^c \times \mathbb{R}^+ \times \mathsf{MC}^c$ as the least multi-relation given by the rules. A particular expression E then gives rise to a Markov chain with initial state E and a discrete state space S, determined by the states reachable from E.

Note that, as before, an expression E like $(\lambda).E' + (\mu).E''$ has two alternatives. But different from Section 2 where the decision which alternative to take has been nondeterministic this is not the case here. Instead, the decision is governed by the probabilistic evolution of $(\lambda).E'$ and $(\mu).E''$, since a race is assumed to exist between the different branches. The sojourn time of E, i.e. the time until the state changes (to either E' or E'') is then exponentially distributed with rate $(\lambda + \mu)$. As a consequence of property (D) of exponential distributions, E' (resp. E'') will win the race with probability $\lambda/\lambda + \mu$ (probability $\mu/\lambda + \mu$).

In other words, imagine we want to add a probabilistic choice operator $\oplus_p$ – that selects its left-hand side with probability p, and it's right-hand side with $1 - p$. We could indeed do so easily, as long as it is guarded by some delay prefix. We could define

$$(\lambda).(E' \oplus_p E'') = ((1-p)\lambda).E' + (p\lambda).E'' \tag{6}$$

to manifest the probabilistic effect that the operator '+' has due to the race condition.

This remark leads us to the algebraic properties of bisimilarity (or lumpability) in this context. From the above discussion, it is obvious that the laws in

(C)	$E + F = F + E$
(A)	$(E + F) + G = E + (F + G)$
(I')	$(\lambda).E + (\mu).E = (\lambda + \mu).E$
(N)	$E + 0 = E$

Table 8. Axioms for MC bisimilarity.

Table 4 cannot be valid in MC^c without change. The idempotence law (I) clearly contradicts the race condition we assume. Instead, a revised law (I') is needed that reflects the minimum property (C) of exponential distributions. It is listed in Table 8 together with the main equational laws characterising lumpability [20].

4 Interactive Markov Chains

This section joins the models of the preceding two sections, continuous time Markov chains and labelled transition system in an orthogonal fashion. It does so by means of two types of prefixes. The action-prefixed expression $\mathsf{a}.P$ may interact on action a and afterwards behave as expression E. The delay-prefixed expression $(\lambda).F$ has to delay for an exponentially distributed time according to rate λ before turning into expression F. We first introduce the language and discuss the equational properties we expect to hold for this language. Then we match the equational theory with a corresponding operational definition (in SOS style) and appropriate notions of semantic equivalences, based on strong bisimulation and weak congruence.

4.1 Algebra of Interactive Markov Chains

Definition 16. *Let* $\mathsf{a} \in \mathcal{A}_\tau$, $\lambda \in \mathbb{R}^+$, *and* $X \in \mathcal{V}$. *We define the language* IMC^c *as the set of* closed *expressions given by the following grammar.*

$$\mathcal{E} ::= \quad 0 \quad | \quad \mathsf{a}.\mathcal{E} \quad | \quad (\lambda).\mathcal{E} \quad | \quad \mathcal{E} + \mathcal{E} \quad | \quad X \quad | \quad [X := \mathcal{E}]_i$$

Example 17. A simple example of an expression of IMC^c is

$$\mathsf{a}.(\lambda).(\mu).\mathsf{a}.(\mu).0$$

Our intuition is as follows: The expression initially is ready to interact on action a. Afterwards, it delays the next possible interaction on a by a sequence of exponential delays, the first given by rate λ, the second given by rate μ. After a second interaction on a it delays for another exponentially distributed duration, before turning into the terminated expression.

This small example of what we intend to do with IMC^c does not cover all possibilities. In general, we can combine delays and actions, such as in

$$\mathsf{a}.P + (\lambda).Q$$

As a consequence, we have to develop an unambiguous view of the interplay of actions and delays. To do so, we discuss the meaning of IMC^c from an algebraic perspective, by stating which expressions of IMC^c can be equated with respect to an intuitive notion of equality. It is important to observe that we have some freedom with respect to what we consider to be intuitive, but there are also constraints.

Strong bisimulation. First of all we intend to inherit the algebras PA^c and IMC^c i.e., the laws we established earlier should remain valid. As a consequence, our equational theory for strong bisimulation for IMC is based on the union of the axioms listed in Table 4 and in Table 8. We could decide that this union is precisely covering all the cases we want to equate with a strong bisimulation. This would mean that the meaning of $\mathsf{a}.P + (\lambda).Q$ has to be obtained somehow by interpreting the class of all algebraically equivalent expressions, i.e. all expressions into which it can be rewritten using the axioms. Here, we decide not to follow this purely algebraic approach, but instead to add one further axiom that corresponds to an operational intuition. This is the notion of *maximal progress*: we assume that intuitively actions can happen as soon as possible. This means that a behaviour such as $\mathsf{a}.P + (\lambda).Q$ will not be delayed at all if the action a is instantaneously enabled. In this case, $\mathsf{a}.P + (\lambda).Q$ will behave just like $\mathsf{a}.P$ (since the probability of the delay to finish is $1 - e^{-\lambda 0} = 0$). But how do we know that action a is indeed enabled? We do *not* know this in general, because the action may depend on some interaction with the environment which is delayed. But in the specific case where action a is the distinguished internal action τ the environment cannot influence its occurrence, and therefore it can occur instantaneously. Hence we can add an axiom

$$(P) \quad (\lambda).E + \tau.F \;=\; \tau.F$$

to our equational theory, reflecting our *maximal progress assumption*. This assumption is often made in real-time process algebra [45,59,52,12].[1]

The resulting set of core axioms for strong bisimilarity on IMC is listed in Table 9. All of them have appeared in the subalgebras of PA^c and MC^c, except for (P) and for the law (I''). The latter restricts the standard idempotence law (I) (i.e., $E + E = E$) to action prefixed expressions, such that it is compatible with (I'). Recall that (I) contradicts the race condition assumed in the MC context, and hence needs a refinement.

[1] In the context of generalised stochastic Petri nets a similar assumption is present: by definition, immediate transition are assumed to have a higher priority level than Markov timed transitions [2].

(C)	$E + F = F + E$
(A)	$(E + F) + G = E + (F + G)$
(I')	$(\lambda).E + (\mu).E = (\lambda + \mu).E$
(I'')	$\mathsf{a}.E + \mathsf{a}.E = \mathsf{a}.E$
(N)	$E + 0 = E$
(P)	$(\lambda).E + \tau.F = \tau.F$

Table 9. Axioms for IMC strong bisimilarity.

(C)	$E + F = F + E$
(A)	$(E + F) + G = E + (F + G)$
(I)	$\mathsf{a}.E + \mathsf{a}.E = \mathsf{a}.E$
(I')	$(\lambda).E + (\mu).E = (\lambda + \mu).E$
(N)	$E + 0 = E$
(P)	$(\lambda).E + \tau.F = \tau.F$
$(\tau 1)$	$\mathsf{a}.\tau.E = \mathsf{a}.E$
$(\tau 1')$	$(\lambda).\tau.E = (\lambda).E$
$(\tau 2)$	$E + \tau.E = \tau.E$
$(\tau 3)$	$\mathsf{a}.(E + \tau.F) + \mathsf{a}.F = \mathsf{a}.(E + \tau.F)$

Table 10. Axioms for IMC weak congruence.

Weak congruence. Even though the axiom (P) of strong bisimilarity involves a specific treatment of internal actions, the axiom system in Table 9 does not provide means to abstract from sequences of internal actions. Recall that Table 5 presents axioms $(\tau 1)$–$(\tau 3)$ that reflect the power of weak congruence to eliminate sequences of internal actions. A similar treatment of internal actions is desirable for IMC, and we therefore postulate some axioms for a weak congruence on IMC. The axioms are listed in Table 10. They extend the ones we have postulated for strong bisimulation on IMC (because we want to preserve the equations of strong bisimulation) with additional axioms that take care of internal actions. The axioms $(\tau 1)$–$(\tau 3)$ are known from the PA^c context already, and axiom $(\tau 1')$ is an obvious adaption of $(\tau 1)$ to the delay prefixed case: if we can do an internal move after some delay, we can also skip the internal move, but not the delay.

Studying the equational theory in Table 10 raises the question why axioms $(\tau 2)$ and $(\tau 3)$ do not require a similar adaptation to the delay-prefix case as $(\tau 1)$ does. For $(\tau 2)$, the answer is easy, because it is not specific for the action-prefix case, it also covers the cases where E involves delay prefixes. But for $(\tau 3)$ the answer is more involved. The adapted candidate law

$$(\tau 3') \quad (\lambda).(E + \tau.F) + (\lambda).F = (\lambda).(E + \tau.F)$$

$$\frac{}{\mathsf{a}.E \xrightarrow{\mathsf{a}} E} \qquad \frac{E \xrightarrow{\mathsf{a}} E'}{E + F \xrightarrow{\mathsf{a}} E'} \qquad \frac{F \xrightarrow{\mathsf{a}} F'}{E + F \xrightarrow{\mathsf{a}} F'} \qquad \frac{E_i\{[X := E]_i / X_i\} \xrightarrow{\mathsf{a}} E'}{[X := E]_i \xrightarrow{\mathsf{a}} E'}$$

$$\frac{}{(\lambda).E \xrightarrow{\lambda} E} \qquad \frac{E \xrightarrow{\lambda} E' \quad F \not\xrightarrow{\tau}}{E + F \xrightarrow{\lambda} E'} \qquad \frac{E \xrightarrow{\lambda} E' \quad F \not\xrightarrow{\tau}}{F + E \xrightarrow{\lambda} E'} \qquad \frac{E_i\{[X := E]_i / X_i\} \xrightarrow{\lambda} E'}{[X := E]_i \xrightarrow{\lambda} E'}$$

Table 11. Operational semantic rules for IMC^c.

is *not* sound for IMC weak congruence, since on the left hand side, the time needed before being able to behave as F is governed by an exponential distribution with rate 2λ, while the process on the right is slower, since it evolves into F after a delay with a rate of only λ. The fact that law $(\tau 3')$ is invalid shades some interesting light on our definition, and suggests a resemblance to the behavioural equivalence known as branching bisimulation [54]. The interested reader is referred to [19] for a detailed discussion concerning this similiarity.

We conclude our an axiomatic view on IMC by pointing out that this perspective still has to be matched by an operational definition of the semantics that matches these axioms – up to appropriate notions of equalities, which also need to be defined.

4.2 Semantics

Interactive Markov Chains involve two types of prefixes. On the semantic level this leads to a model with a twofold transition relation $\longrightarrow$ and $-\!\!\rightarrow$. The former represents action transitions, the latter represents Markov transitions. This should not be surprising, since we strive for an orthogonal extension of PA^c and MC^c.

To give a meaning to elements of IMC^c we define an operational semantics on the basis of the SOS-rules introduced for PA^c (Table 2) and MC^c (Table 7), except for one change.

Definition 17. *The* action transition *relation* $\longrightarrow \subset \mathsf{IMC}^c \times \mathcal{A}_\tau \times \mathsf{IMC}^c$ *is the least relation and the* Markov transition *relation* $-\!\!\rightarrow \subset \mathsf{IMC}^c \times \mathbb{R}^+ \times \mathsf{IMC}^c$ *is the least multi relation given by the rules in Table 11, where* $E \not\xrightarrow{\tau}$ *denotes that there is no* $E' \in \mathsf{IMC}^c$ *such that* $E \xrightarrow{\tau} E'$.

Compared to the rules in Table 2 and, Table 7, two rules are now equipped with negative premises of the form $E \not\xrightarrow{\tau}$ meaning that no internal transition can be performed by E. Only in this case, a Markov transition can happen in the context of choice. This negative premise[2] is used to encode *maximal progress*:

[2] The use of a negative premise is not mandatory to make the above axiom system sound. Alternatively, one can take the operational rules as the plain union of the ones

An expression is only allowed to delay, if it has nothing internal to do instantaneously. Note that the additional negative premise is only influencing the behaviour of expressions that involve both delay prefixes and action prefixes. So, if restricted to the sublanguages PA^c and MC^c, the operational semantics reduces to the ones introduced in Table 2 and Table 7.

Example 18. The semantics of the process E_{65} defined by $(2\lambda).(\tau.0 + \mathsf{a}.0)$ is depicted in the upper left of Figure 10. As another example, the semantics of the process E_{66} defined by $[X_1 := \tau.X_2\ ,\ X_2 := \tau.X_1 + (2\lambda).(\tau.0 + \mathsf{a}, 0)]_1$ is depicted in the upper right of the figure.

4.3 Equivalence

We now investigate how equivalences on IMC^c can be defined. Action transitions and Markov transitions coexist in IMC. Meaningful equivalences for IMC should thus reflect their coexistence. Strong and weak bisimilarities will therefore be based on the respective notions for PA^c and MC^c. Additionally, the interrelation of action and Markov transitions has to be captured as well, according to the axioms we have postulated in Section 4.1.

Strong bisimulation. We introduce strong bisimilarity based on Definition 7 and 13.

Definition 18. *An equivalence relation $\mathcal{E}$ on IMC^c is a strong bisimulation iff $P\ \mathcal{E}\ Q$ implies for all $\mathsf{a} \in \mathcal{A}_\tau$*

1. *$P \xrightarrow{\mathsf{a}} P'$ implies $Q \xrightarrow{\mathsf{a}} Q'$ for some Q' with $P'\ \mathcal{E}\ Q'$,*
2. *$\gamma_{\mathrm{M}}(P, C) \leq \gamma_{\mathrm{M}}(Q, C)$ for all equivalence classes C of $\mathcal{E}$.*

Two processes P and Q are strongly bisimilar (written $P \sim Q$) if they are contained in some strong bisimulation.

This definition amalgamates strong bisimilarity for PA^c and for MC^c. In order to compare the stochastic timing behaviour, the cumulative rate function γ_{M} is used, as motivated in Section 3.2. Formally speaking bisimilarity conservatively extends [13] the respective notions on basic process algebra processes and Markov chains. This answers indeed why we have reused the symbol $\sim$, that has been used in Section 2.4 already to denote strong bisimilarity on PA^c.

We obtain the following desirable results:

for PA^c and MC^c, as done in [20]. In this case a more involved definition of strong and weak bisimulation is needed that must now incorporate maximal progress. The way we proceed here is sketched in [19, Sec. 6.1] and elaborated in [7]. The solution is didactically more appealing, but has the drawback that divergence may imply the awkward phenomeon of a time deadlock: If a state is on a cycle of internal transitions, this implies that no Markov transition (indicating time progress) can be derived, even though the system may return to this state via the internal steps ad infinitum.

Proposition 3. *Strong bisimilarity is*

- *an equivalence relation on* IMC^c.
- *a strong bisimulation on* IMC^c.
- *the* largest *strong bisimulation on* IMC^c.

In addition, strong bisimulation turns out to be the desired notion of equivalence relative to the equational theory we postulated.

Theorem 4. *Strong bisimilarity is a congruence relation with respect to the operators of* PA^c*, and it satisfies the axioms in Table 9.*

By adding additional laws to handle recursion the equational theory induced by Table 9 can be shown to completey characterise strong bisimulation, see [19].

Weak congruence. Strong bisimilarity does not abstract from sequences of internal transitions like *weak* bisimilarity does (cf. Section 2.4). We will therefore try to find a corresponding definition of a weak relation for IMC, that is, a weak relation that complies to the axioms we have postulated in Table 10.

A few questions have to be addressed in order to define weak bisimulation on IMC properly. While the treatment of action transitions can follow the lines of Section 2.4, the treatment of Markov transitions in a weak bisimulation has to be clarified. As remarked in Section 3.2 it is impossible to replace a sequence of Markov transitions by a single Markov transition without affecting the probability distribution of the total delay. So, we are forced to demand that Markov transitions have to be bisimulated in the *strong* sense, using function γ_{M}, even for a weak bisimulation. However, we allow them to be preceded and followed by arbitrary sequences of internal action transitions. These sequences are, according to Definition 9 given by $\stackrel{\varepsilon}{\Longrightarrow}$, the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$. To incorporate these sequences into the definition of weak bisimulation is a bit involved technically. For strong bisimilarity, γ_{M} has been used to cumulate rates of Markov transitions that directly lead from a state P into a specific equivalence class C. We broaden this treatment in order to keep track of the impact of internal transitions that *follow* a Markov transition: We cumulate all rates of Markov transitions leading to states that can internally evolve into an element of class C. For this purpose, we define the *internal backward closure* C^τ as the set of processes that may internally evolve into an element of a set C, i.e. $C^\tau = \{P' \mid \exists P \in C : P' \stackrel{\varepsilon}{\Longrightarrow} P\}$.

Example 19. Concerning the IMC E_{67} in Figure 10, the internal backward closure 

of the set  is the union of the sets  and  . 

is  and 

is  .

The treatment of internal sequences *preceding* a Markov transitions can follow the style of Definition 10. Thus, whenever there is a sequence of internal transitions to some state P, the cumulative rate $\gamma(P, C^\tau)$ should be taken into account for comparison purposes. This requirement will be made more precise in the following definition.

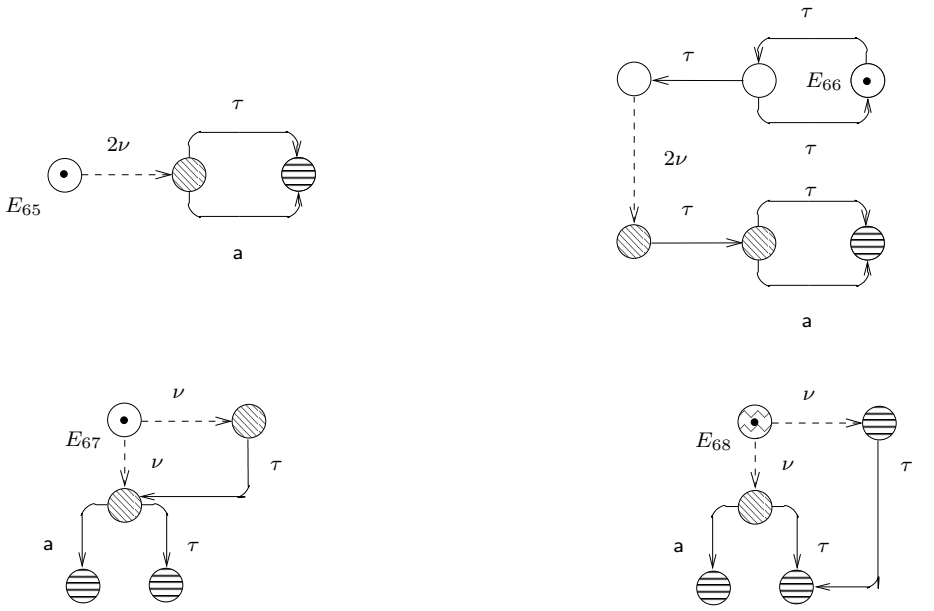


Fig. 10. Some characteristic examples for weak bisimilarity.

Definition 19. *An equivalence relation $\mathcal{E}$ on IMC^c is a weak bisimulation iff $P\ \mathcal{E}\ Q$ implies for all $\mathsf{a} \in \mathcal{A}_\varepsilon$*

1. *$P \stackrel{\mathsf{a}}{\Longrightarrow} P'$ implies $Q \stackrel{\mathsf{a}}{\Longrightarrow} Q'$ for some Q' with $P'\ \mathcal{E}\ Q'$,*
2. *$P \stackrel{\varepsilon}{\Longrightarrow} P'$ imply $Q \stackrel{\varepsilon}{\Longrightarrow} Q'$ for some Q' with $\gamma_{\mathrm{M}}(P', C^\tau) \leq \gamma_{\mathrm{M}}(Q', C^\tau)$ for all equivalence classes C of $\mathcal{E}$.*

Two processes P and Q are weakly bisimilar (written $P \approx Q$) if they are contained in some weak bisimulation.

We illustrate the distinguishing power of $\approx$ by means of some examples.

Example 20. E_{65} and E_{66} depicted in Figure 10 are equivalent even though the precise argument is somewhat involved. We have for E_{65} that $\gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\circledcirc}) = 2\nu$, as well as $\gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\circledcirc}) = 2\nu$. E_{66} has value 0 for both classes, but this does not violate the conditions imposed by clause (2) of Definition 19. Instead, we have to find a state reachable via τ inside class $\bigcirc$ satisfying $\gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\circledcirc}) \geq 2\nu \leq \gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\circledcirc})$. Indeed we get the values 2ν precisely for the leftmost state $\bigcirc$ reachable from E_{66}, and hence clause (2) of Definition 19 is satisfied. On the other hand, also the rate of E_{66} has to be investigated. We see that for E_{66}, $\gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\circledcirc}) = 0 = \gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\circledcirc})$, which is at most the values of E_{65}. Note that in this reasoning, it is important that we do not demand equality of cumulated rates in clause (2), but instead demand to find a matching

state with *at least* ($\leq$) the cumulated rates of the state we have to check. Hence $E_{65} \approx E_{66}$.

The process E_{67} is equivalent to the former two, because $\gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\diagup\!\!\diagup}) = 2\nu$ and $\gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\diagup\!\!\diagup}) = 2\nu$. In contrast, $\gamma_{\mathrm{M}}(\bigcirc, \overset{\tau}{\diagup\!\!\diagup}) = \nu$ whence we have that E_{68} is not weakly bisimilar to the former three processes.

We get the following desirable properties of $\approx$.

Proposition 4. *Weak bisimilarity is*

- *an equivalence relation on* IMC^c.
- *a weak bisimulation on* IMC^c.
- *the* largest *weak bisimulation on* IMC^c.
- *a congruence with respect to all operators of* IMC^c except the choice operator '$+$'.

The fact that weak bisimilarity is not substitutive with respect to choice is inherited from non-stochastic weak bisimilarity, cf. Section 2.4. In order to rectify this situation we identify a specific congruence contained in $\approx$, along the lines of Definition 11.

Definition 20. *P and Q are weakly congruent, written $P \simeq Q$, iff for all $\mathsf{a} \in \mathcal{A}_\tau$ and all $C \in \mathsf{IMC}^c / \approx$:*

1. $P \xrightarrow{\mathsf{a}} P'$ *implies* $Q \overset{\varepsilon}{\Longrightarrow} \xrightarrow{\mathsf{a}} \overset{\varepsilon}{\Longrightarrow} Q$ *for some* Q' *with* $P' \approx Q'$,
2. $Q \xrightarrow{\mathsf{a}} Q'$ *implies* $P \overset{\varepsilon}{\Longrightarrow} \xrightarrow{\mathsf{a}} \overset{\varepsilon}{\Longrightarrow} P'$ *for some* P' *with* $P' \approx Q'$,
3. $\gamma_{\mathrm{M}}(P, C) = \gamma_{\mathrm{M}}(Q, C)$,

Weak congruence strengthens the requirement of weak bisimilarity of initial internal transitions in precisely the same way as in Definition 11. It requires that an initial internal transition has to be matched by at least one internal transition. This small change it is again sufficient to fix the congruence problem with respect to choice: weak congruence is a proper substitutive relation with respect to all language operators.

Theorem 5. *Weak congruence is a congruence with respect to all operators of* IMC^c*, and it satisfies the axioms in Table 10.*

By adding further laws the equational theory induced by Table 9 can be shown to be sound and complete with respect to weak congruence on IMC^c [19,17] . Furthermore, weak congruence is the *coarsest* congruence contained in weak bisimilarity, as a consequence of the following lemma (cf. Lemma 3).

Lemma 5. *$E_1 \simeq E_2$ iff, for each $E_3 \in \mathsf{IMC}^c$, $E_1 + E_3 \approx E_2 + E_3$ and $E_3 + E_1 \approx E_3 + E_2$.*

As a result, we have obtained two substitutive equivalence notions on IMC^c: strong bisimilarity and weak congruence, a distinguished subset of weak bisimilarity. The interrelation between these equivalences is expressed in the following lemma (cf. Lemma 4).

$$\frac{P \xrightarrow{\lambda} P' \qquad Q \not\xrightarrow{\tau}}{P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q \xrightarrow{\lambda} P' \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q} \qquad \frac{Q \xrightarrow{\lambda} Q' \qquad P \not\xrightarrow{\tau}}{P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q \xrightarrow{\lambda} P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q'}$$

$$\frac{P \xrightarrow{\lambda} P' \qquad \texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P \not\xrightarrow{\tau}}{\texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P \xrightarrow{\lambda} \texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P'}$$

Table 12. Structural operational rules for parallel composition and abstraction.

Lemma 6. $\sim \subset \simeq \subset \approx$.

Summarizing, we have managed to integrate basic process algebra – where strong bisimulation and weak congruence are reference notions – and Markov chain algebra – where lumpability is central – in a single algebra. An obvious next step now is to extend this algebraic core with the other operators from process algebra to enable the concurrent composition of IMC expressions and abstraction from observable actions.

4.4 Concurrency and Abstraction

The two operators for abstraction and parallel composition, cf. Section 4, can be added to IMC^c without disturbing any of the theory. We extend IMC^c with these operators by stipulating that if P and Q are in IMC^c then $P \parallel \mathsf{a}_1 \ldots \mathsf{a}_n \parallel Q$ and $\texttt{hide } \mathsf{a}_1 \ldots \mathsf{a}_n \texttt{ in } P$ are is in IMC^c as well. The semantics of these operators are given by the operational rules in Table 2, Table 3, and Table 12.

According to this definition, action transitions are treated precisely as in the PA^c setting. Markov transitions $\longrightarrow$ are only possible if maximal progress is assured, which is incorporated via negative premises. Negative premises in such rule schemata have to be treated carefully in general, since they may affect the well-definedness of the induced transition relation [17]. In this case, however, it is not difficult to show that the rule schemata are well-defined.

In the case of parallel composition, it should be noted that the Markovian delay transitions are interleaved as if they were standard action transitions, in particular without adjusting rates. This is a consequence of the memoryless property (cf. property **B** on page 204), and one of the principal reasons why exponential distributions fit so well to process algebra. In the following Section 5 we will elaborate on the appropriateness of this combination.

One of the consequences of this independent delaying is that the expansion law (X) (cf. Table 6) can be extended in a rather straightforward way to Interactive Markov Chains. Table 13 lists the resulting law, together with an additional law for abstraction, that (together with the ones in Table 6) allow one to rewrite parallel composition, as well as abstraction into the basic operators of IMC^c.

Example 21. In order to exercise the modelling of concurrent behaviour with IMC we consider two processes, E_{71} and E_{72}, depicted in Figure 11. They are

$$(X') \quad P \,|[\, \mathtt{a}_1 \ldots \mathtt{a}_n \,]|\, Q = \sum(\lambda_i).(P_i \,|[\, \mathtt{a}_1 \ldots \mathtt{a}_n \,]|\, Q) + \sum_{\mathtt{a}_j \notin \{\mathtt{a}_1 \ldots \mathtt{a}_n\}} \mathtt{a}_j.(P_j \,|[\, \mathtt{a}_1 \ldots \mathtt{a}_n \,]|\, Q) +$$

$$\overbrace{\underbrace{\sum(\lambda_i).P_i + \sum \mathtt{a}_j.P_j}_{P} \quad \underbrace{\sum(\mu_k).Q_k + \sum \mathtt{b}_l.Q_l}_{Q}}^{} \qquad \sum(\mu_k).(P \,|[\, \mathtt{a}_1 \ldots \mathtt{a}_n \,]|\, Q_k) + \sum_{\mathtt{b}_l \notin \{\mathtt{a}_1 \ldots \mathtt{a}_n\}} \mathtt{b}_l.(P \,|[\, \mathtt{a}_1 \ldots \mathtt{a}_n \,]|\, Q_l) +$$

$$\sum_{\mathtt{a}_j = \mathtt{b}_l \in \{\mathtt{a}_1 \ldots \mathtt{a}_n\}} \mathtt{a}_j.(P_j \,|[\, \mathtt{a}_1 \ldots \mathtt{a}_n \,]|\, Q_l)$$

$$(H2') \quad \mathtt{hide}\ \mathtt{a}_1 \ldots \mathtt{a}_n\ \mathtt{in}\ \ (\lambda).P = (\lambda).\mathtt{hide}\ \mathtt{a}_1 \ldots \mathtt{a}_n\ \mathtt{in}\ \ P$$

Table 13. Axioms for rewriting parallel composition and abstraction on IMC^c.

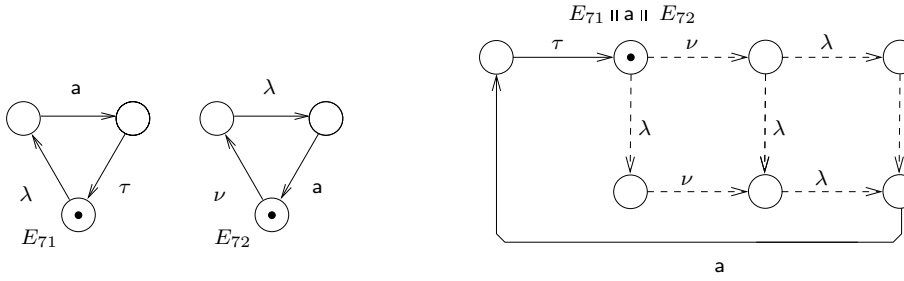


Fig. 11. Parallel composition of IMC.

defined by $[X_1 := (\lambda).\mathtt{a}.\tau.X_1]_1$, respectively $[X_1 := (\nu).(\lambda).\mathtt{a}.X_1\ )]_1$. Their parallel composition $E_{71} \,|[\, \mathtt{a} \,]|\, E_{72}$ is depicted on the right of the figure. Note that maximal progress enforces the τ-transition of E_{71} to take precedence over a delay (with rate ν) of E_{72} prior to reeintering the initial state.

Figure 12 illustrates the semantics of ($\mathtt{hide\ a\ in}\ \ E_{71} \,|[\, \mathtt{a} \,]|\, E_{72}$), a process where all actions are internalised. In this figure, states shaded with equal patterns are weakly congruent. The shading indicates that this process is weakly congruent to a process E_{73} defined by $[X_1 := (\nu).(2\lambda).(\lambda).X_1 + (\lambda).(\lambda).(\nu).X_1\ )]_1$. This is a small Markov chain depicted on the right of Figure 12. The fact that

$$(\mathtt{hide\ a\ in}\ \ E_{71} \,|[\, \mathtt{a} \,]|\, E_{72}\,) \ \simeq\ E_{73},$$

means that the concurrent execution of the Interactive Markov Chains E_{71} and E_{72} can be concisely represented by the Markov chain E_{73}. In other words, we have generated a small Markov chain from the composition of two IMCs.

This is a very simple example showing how Markov chains can be compositionally specified with IMC. Subsequently, the model can be evaluated with standard analysis techniques for Markov chains, cf. [18]. A much larger case study is developed in [22], where a Markov chain model of 720 states is derived from

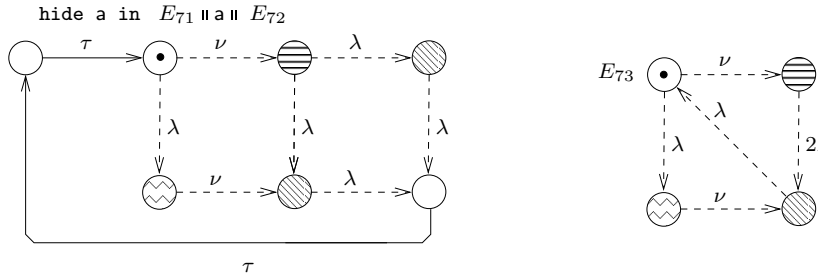


Fig. 12. Compositional specification of a Markov chain.

an IMC specification of the plain old telephone system involving more than 10^7 states. To circumvent the state space explosion problem, the case study makes heavy use of substitutivity (i.e., congruence) properties and algorithms for simplifying (i.e., lumping) the state space (of components) according to weak congruence. We refer to [19] for further reading on IMC.

5 Related Work: A Comparison of Algebraic Principles

As was already mentioned in the introduction a fair number of stochastic process algebras have been developed during the last decade or so, IMC being one of them. In this section we want to make a comparison between them. We will not do so in terms of the complete formalisms, but will organise our discussion around the potential resolution strategies for a number of issues that arise inevitably when trying to combine well-known process algebraic principles with the features of continuous time Markov chains. This will give, we hope, a more generic insight into the (im)possibilities of the various approaches.

We will forego the challenge of defining an integrating semantical model for the various formalisms for a deeper mathematical comparison of the various constructs. Instead, we will try and make our comparison in terms of the various algebraic principles, i.e. in terms of the sort of equational laws that are involved. We think that this is the best level of abstraction to discuss the different options and their consequences.

The three central questions that must be addressed when developing Markovian process algebras are:

1. the meaning of choice
2. the meaning of concurrent composition
3. the meaning of synchronisation

We address these questions in the following sections. Our notational vehicle will be slightly different from that of IMC, because the other stochastic process algebras do not have the separation between actions and delays. We will therefore have only one action-prefix construct, viz.

$$(\mathsf{a}, \lambda).B$$

meaning that action a may take place after an exponentially distributed delay with rate λ, resulting in the behaviour specified by B. As we will also want to discuss some aspects stochastic process algebra in a non-Markovian setting, we also introduce the notation

$$(\mathsf{a}, F).B$$

where F is denotes a (general) distribution of the delay associated with a. We allow F to be denoted by an expression over one or more stochastic variables whose distributions are (implicitly) given, e.g. $(\mathsf{a}, X).B$ implies that a is delayed with the distribution of X, and $(\mathsf{a}, f(X, Y)).B$ means that a is delayed with the distribution defined by $f(X, Y)$ for given distributions of X and Y.

5.1 The Meaning of Choice

The choice or summation operator affects the branching structure of the transition system that is described: its SOS style semantics yields multiple outgoing transitions from a state in the underlying transition system. In CTMCs such outgoing transitions are interpreted as creating a *race condition*, i.e. they are seen as processes that are competing for the fastest service according to their distributions. As we have seen, for exponential distributions the time until the first transition fires is again exponentially distributed, with a rate that is the sum of the rates of the individual transitions.

It stands to reason that in a Markovian process algebra we should somehow be able to add up the rates of all transitions with identical action labels. Indeed, such transitions will all be in a race condition when the environment has enabled the corresponding action. Using the probabilistic choice operator $\oplus_p$ introduced in Section 3.3 we can write this down as an algebraic law, the *race condition principle* (RCP):

$$(\mathsf{a}, \lambda).B + (\mathsf{a}, \mu).C \;=\; (\mathsf{a}, \lambda + \mu).(B \oplus_{\frac{\mu}{\lambda+\mu}} C) \tag{7}$$

Note that the right-hand side of the equation can be interpreted as a process for which an a-transition with the combined rates leads to a *superposition state* that, when it is reached, reduces instantaneously to one of its constituents states. This occurs with a probability proportional to the relative weight of the corresponding rate.

This principle can also be formalised for the non-Markovian case, where a race condition between arbitrary continuous distributions takes place (e.g. as in semi-Markov chains):

$$(\mathsf{a}, X).B + (\mathsf{a}, Y).C \;=\; (\mathsf{a}, \min(X, Y)).(B \oplus_{P\{Y<X\}} C) \tag{8}$$

The above laws make it clear that in a stochastic setting the choice operator has to take the *capacity* of its arguments into account. This is even clearer in the case of IMC, where the delay operator can be interpreted as a scalar multiplication w.r.t. choice:

$$(\lambda).B + (\mu).B \quad = \quad (\lambda + \mu).B \tag{9}$$

This is in stark contrast to the usual interpretation of the choice operator in process algebra, which could be referred to as *structural*, in the sense that only of the arguments is chosen, and therefore there is no interference with properties of the conflicting transitions. This leads to the *idempotency* law for choice, which could be seen as a kind of 'poor man's choice': choosing between identical arguments is as good as no choice at all:

$$B + B \quad = \quad B \tag{10}$$

The difference between the laws (7) and (10) is very important, as the choice operator plays a crucial role in the formulation of other laws in process algebra, the so-called *expansion laws* in particular. Below we will discuss the implications that the capacitive interpretation of choice in the context of parallel composition.

It is possible to interpret (10) as a limit case of (7) by interpreting the former as the behaviour for *immediate* transitions, cf. [27]. They can be thought of as having an infinite rate ∞, with the property $\infty + \infty = \infty$. This approach makes (10) compatible with (7) for actions for which this is a reasonable assumption (e.g. τ-actions).

The process algebra EMPA [4] also wants to apply (10) to so-called *passive* actions, actions that have no associated (finite) rates, but obtain them by synchronising with non-passive actions. This, however, leads to complications, as we will see below.

5.2 Concurrent Composition

The next operator that we consider is the parallel or concurrent composition of processes. As we will consider the issue of synchronisation of actions separately, here we concentrate on 'pure' *interleaving*, i.e. parallel composition without synchronisation of actions between components.

To guide our discussion we consider the following, simple expansion law of standard process algebra:

$$\mathsf{b}.B \,|||\, \mathsf{c}.C \quad = \quad \mathsf{b}.(B \,|||\, \mathsf{c}.C) + \mathsf{c}.(\mathsf{b}.B \,|||\, C) \tag{11}$$

The first thing that we can observe is that for general distributions the interleaving law does not hold, i.e.

$$(\mathsf{b}, X).B \,|||\, (\mathsf{c}, Y).C \quad \neq \quad (b, X).(B \,|||\, (\mathsf{c}, Y).C) + (\mathsf{c}, Y).((\mathsf{b}, X).B \,|||\, C) \tag{12}$$

because the occurrence of b after c has taken time to occur generally has another distribution than b occurring initially.

One way to deal with this complication is to restrict to exponential distributions, i.e. the Markovian case. Because of the memoryless property of these distributions, they are perfectly compatible with the interleaving law as distributions are not affected by the conditional information that one action takes place only after the occurrence of another. So we have

$$(\mathsf{b},\lambda).B \;|||\; (\mathsf{c},\mu).C \;=\; (\mathsf{b},\lambda).(B \;|||\; (\mathsf{c},\mu).C) + (\mathsf{c},\mu).((\mathsf{b},\lambda).B \;|||\; C) \qquad (13)$$

showing again the perfect match of interleaving process algebra and continuous time Markov chains, as was already evident from the elegant theory of IMC.

It is worthwhile to consider also ways out of the complications of (12), as for many applications the assumption of memorylessness is too strong. One straightforward way is to complicate the interleaving law by adding the conditional information that was missing. In this way we obtain:

$$\begin{aligned}(\mathsf{b},X).B \;|||\; (\mathsf{c},Y).C \;=\;& (\mathsf{b},X).(B \;|||\; (\mathsf{c},\langle Y-X|X<Y\rangle).C) + \\ & (\mathsf{c},Y).((\mathsf{b},\langle X-Y|X>Y\rangle).B \;|||\; C)\end{aligned} \qquad (14)$$

where $\langle Y-X|X<Y\rangle$ denotes the distribution of $Y-X$ under the condition that $X<Y$.

The disadvantage of this approach is that the formula complicates very rapidly if more than two actions interleave with nested conditional distributions. This is unattractive, not only algebraically, but also in the sense that the calculation of such complicated conditional distributions is computationally expensive, e.g. when doing simulations. Presumably because of this reasons this approach has not been pursued in (non-Markovian) process algebra.

A second way out is provided by following the separation of concerns between delay transitions and actions, as was also done in IMC. If this idea is combined with the use of stochastic *clocks* to guard the occurrence of transitions, we can formulate interleaving again in terms of an elegant algebraic law, even if one considers the non-Markovian case. Let $\{X_1,\ldots,X_n\}B$ mean that in the initial state of B the clocks $X_1,\ldots,X_n$ are set with random samples according to their associated distributions over $\mathbb{R}^+$, after which they start counting down until they expire (reach 0). Let $(X \to \mathsf{b})$ mean that b is delayed until X has expired. With these additional stochastic clock operators we now get a new interleaving law, viz.

$$\begin{aligned}&\{X,Y\}(X \to \mathsf{b}).B \;|||\; (Y \to \mathsf{c}).C \;= \\ &\qquad \{X,Y\}((X \to \mathsf{b}).(B \;|||\; (Y \to \mathsf{c}).C) + (Y \to \mathsf{c}).((X \to \mathsf{b}).B \;|||\; C))\end{aligned} \qquad (15)$$

where we see that the guarded actions $(X \to \mathsf{b})$ and $(Y \to \mathsf{c})$ are interleaved, but the clock setting $\{X,Y\}$ is not. This approach to non-Markovian process algebra is elaborated in [33].

A final approach that we wish to mention in connection with the interleaving law in a stochastic context is to give up the law altogether. This idea belongs to

the semantic school of the 'true concurrency', which insists that parallel composition is fundamentally different from interleaving and that the two should not be equated. The causal dependencies on the left-hand and right-hand sides of the interleaving law are different (b and c are independent versus b causes/precedes c or vice versa) [36]. It is possible to extend so-called *partial-order semantics* for process algebra with stochastic information to obtain suitable non-interleaving semantical models for stochastic process algebras. We refer to [8,32] for further reading.

5.3 Synchronisation

Probably the most intriguing question in the design of stochastic process algebra is related to the synchronisation of stochastic actions of two concurrent system components. If both actions are subject to stochastic delays, what should be the stochastic delay of their synchronised occurrence?

Looking at this question in its simplest process algebraic form, we consider standard process algebraic law

$$\mathsf{a}.B \parallel \mathsf{a} \parallel \mathsf{a}.C = \mathsf{a}.(B \parallel \mathsf{a} \parallel C) \tag{16}$$

and wonder what are reasonable functions '$*$' that would make the corresponding stochastic equation hold true:

$$(\mathsf{a}, X).B \parallel \mathsf{a} \parallel (\mathsf{a}, Y).C = (\mathsf{a}, X * Y).(B \parallel \mathsf{a} \parallel C) \tag{17}$$

From a stochastic point of view, one may immediately think of various operationalisations of '$*$', such as the maximum of the distributions, or their convolution, minimum, average etc. Interestingly enough, however, the algebraic properties of the involved operators already impose certain restrictions on '$*$' by themselves.

Let us consider the term $((\mathsf{a}, X).B + (\mathsf{a}, Y).C) \parallel \mathsf{a} \parallel (\mathsf{a}, Z).D$. The interplay of choice and synchronisation allows us to derive:

$$\begin{aligned}
((\mathsf{a}, X).B + (\mathsf{a}, Y).C) \parallel \mathsf{a} \parallel (\mathsf{a}, Z).D &= && \text{(by (8))} \\
(\mathsf{a}, \min(X, Y)).(B \oplus_{P\{X>Y\}} C) \parallel \mathsf{a} \parallel (\mathsf{a}, Z).D &= && \text{(by (17))} \\
(\mathsf{a}, \min(X, Y) * Z).((B \oplus_{P\{X>Y\}} C) \parallel \mathsf{a} \parallel D) &= && \text{(distributing } \oplus\text{)} \\
(\mathsf{a}, \min(X, Y) * Z).((B \parallel \mathsf{a} \parallel D) \oplus_{P\{X>Y\}} (C \parallel \mathsf{a} \parallel D)) && && (18)
\end{aligned}$$

On the other hand, by assuming that we have a (classical) expansion law for synchronisation of the form

$$\begin{aligned}
&((\mathsf{a}, X).B + (\mathsf{a}, Y).C) \parallel \mathsf{a} \parallel (\mathsf{a}, Z).D = \\
&\qquad (\mathsf{a}, X * Z).(B \parallel \mathsf{a} \parallel D) + (\mathsf{a}, Y * Z).(C \parallel \mathsf{a} \parallel D)
\end{aligned} \tag{19}$$

we obtain by applying the RCE (8) to the right-hand side:

$$(\mathsf{a}, \min(X * Z, Y * Z)).((B \parallel \mathsf{a} \parallel D) \oplus_{P\{X*Z>Y*Z\}} (C \parallel \mathsf{a} \parallel D)) \tag{20}$$

By equating the terms of (18) and (20) we can conclude that synchronisation with expansion in the context of RCE necessarily leads to the following two requirements:

$$\min(X, Y) * Z = \min(X * Z, Y * Z) \tag{21}$$
$$P\{X < Y\} = P\{X * Z < Y * Z\} \tag{22}$$

In the face of these requirements, it is interesting to see how the main Markovian process calculi have dealt with them.

PEPA. This stochastic process algebra [27] deals with the situation by rejecting classical expansions like (19). To synthesise choice and synchronisation PEPA takes its recourse to so-called *apparent rates*, which replace the original rates when expanding. Interestingly enough, these rates can be obtained in a general setting by combining RCE with (17) only:

$$\begin{aligned}
((\mathsf{a}, \lambda).B + (\mathsf{a}, \mu).C) \,|\!|_{\mathsf{a}}\!|\, (\mathsf{a}, \nu).D = \quad & \text{(by (7))} \\
(\mathsf{a}, \lambda + \mu).(B \oplus_{\frac{\mu}{\lambda+\mu}} C) \,|\!|_{\mathsf{a}}\!|\, (\mathsf{a}, \nu).D = \quad & \text{(by (17))} \\
(\mathsf{a}, (\lambda + \mu) * \nu).((B \,|\!|_{\mathsf{a}}\!|\, D) \oplus_{\frac{\mu}{\lambda+\mu}} (C \,|\!|_{\mathsf{a}}\!|\, D)) = \quad & \text{(by (7))} \\
\left(\mathsf{a}, \frac{\lambda}{\lambda+\mu}((\lambda + \mu) * \nu)\right).(B \,|\!|_{\mathsf{a}}\!|\, D) + \left(\mathsf{a}, \frac{\mu}{\lambda+\mu}((\lambda + \mu) * \nu)\right).(C \,|\!|_{\mathsf{a}}\!|\, D) \quad & (23)
\end{aligned}$$

The two rate parameters occurring in (23) correspond to Hillston's apparent rates. Note that here, however, they are actually independent of the particular synchronisation function '$*$' that is used. Hillston instantiates '$*$' to the *minimum* of rates (corresponding to the distribution of the *slowest* process): for ν greater than $\lambda + \mu$ we obtain:

$$((\mathsf{a}, \lambda).B + (\mathsf{a}, \mu).C) \,|\!|_{\mathsf{a}}\!|\, (\mathsf{a}, \nu).D = (\mathsf{a}, \lambda).(B \,|\!|_{\mathsf{a}}\!|\, D) + (\mathsf{a}, \mu).(C \,|\!|_{\mathsf{a}}\!|\, D).$$

In the converse case that $\nu < \lambda + \mu$ we get:

$$\begin{aligned}
&((\mathsf{a}, \lambda).B + (\mathsf{a}, \mu).C) \,|\!|_{\mathsf{a}}\!|\, (\mathsf{a}, \nu).D = \\
&\qquad \left(\mathsf{a}, \frac{\lambda\nu}{\lambda+\mu}\right).(B \,|\!|_{\mathsf{a}}\!|\, D) + \left(\mathsf{a}, \frac{\mu\nu}{\lambda+\mu}\right).(C \,|\!|_{\mathsf{a}}\!|\, D).
\end{aligned}$$

TIPP. The requirements (21) and (22) can be reformulated for the Markovian case in terms of rates, where we assume that '$*$' is a function over *rates*, as they uniquely determine exponential distributions. We get:

$$(\lambda + \mu) * \nu = (\lambda * \nu) + (\mu * \nu) \tag{24}$$
$$\frac{\lambda}{\lambda + \mu} = \frac{\lambda * \nu}{(\lambda + \mu) * \nu} \tag{25}$$

An obvious solution to fulfil these requirements is followed in the TIPP algebra [23,21], where '$*$' is interpreted as ordinary multiplication. Although this is a very simple and computationally attractive solution for the synchronisation operator, the operational intuition behind this choice is not at all obvious. There is no useful stochastic interpretation of the multiplication of rates that corresponds to some abstract mechanism for synchronisation. In TIPP, therefore, the interpretation of '$*$' is only motivated by its algebraic simplicity.

Buchholz, who essentially adopts this solution too [11], has given a sophisticated twist to the idea to make it more acceptable. For each action a he stipulates the existence of a systemwide (reference) rate μ_{a}. His action-prefix operators then have the format $(\mathsf{a}, r).B$ where the rate of the associated transition is defined as $r.\mu_{\mathsf{a}}$. In this way r defines the relative capacity of a component w.r.t. an action occurrence. At synchronisation these relative capacities are multiplied, e.g. $(\mathsf{a}, 2) * (\mathsf{a}, 0.5) = (\mathsf{a}, 1)$.

Although, the multiplication idea in most of its forms remains questionable as an operational interpretation of synchronisation, it is attractive from the point of view of system decomposition. When we want to decompose a complicated system into a set of simpler systems, then this may be useful from an analytical point of view, even if its does not have a direct operational (or architectural) interpretation. In much of the work centred around (Kronecker) *product forms*, this approach is therefore, often implicitly, followed [50,15].

EMPA. The stochastic process algebra EMPA [4] deals with synchronisation by imposing some restrictions. It starts from an operational interpretation of synchronisation, viz. that all synchronisations take place in a client/server model, where several clients may request a service represented by synchronisation on a given action and one server grants such requests. In such a setup it is reasonable to assume that the server determines the rate of service, and that the clients are 'passive' in this respect.

Algebraically this can be modelled by assuming that all clients have infinite rates ∞ (in principle they are willing to be served instantaneously), and that synchronisation is interpreted as selecting the minimal rate (i.e. the rate of the slowest process), which ultimately means selecting the rate of the server. In formula's we get:

$$\infty * \infty = \infty \tag{26}$$

$$\lambda * \infty = \infty * \lambda = \lambda \tag{27}$$

Assuming at most one synchronising action carries a rate parameter different from ∞, these properties are consistent with (24) and (25). In this way we obtain expansion laws similar to (19):

$$((\mathsf{a}, \lambda).B + (\mathsf{a}, \mu).C) \parallel \mathsf{a} \parallel (\mathsf{a}, \infty).D = (\mathsf{a}, \lambda).(B \parallel \mathsf{a} \parallel C) + (\mathsf{a}, \mu).(B \parallel \mathsf{a} \parallel C) \tag{28}$$

On the other hand, when applying this principle to multiple passive actions the EMPA approach is not free of complications, as it is unclear how the rate ν should be 'distributed' over the passive components. The naive solution with classical expansion does not work here if one also insists on having the idempotency law (10) for passive actions as EMPA does, e.g.

$$\begin{aligned}
&(\mathsf{a}, \lambda).(B \parallel \mathsf{a} \parallel C) \\
&\quad = (\mathsf{a}, \infty).B \parallel \mathsf{a} \parallel (\mathsf{a}, \lambda).C \\
&\quad = ((\mathsf{a}, \infty).B + (\mathsf{a}, \infty).B) \parallel \mathsf{a} \parallel (\mathsf{a}, \lambda).B \\
&\quad = ((\mathsf{a}, \infty).B \parallel \mathsf{a} \parallel (\mathsf{a}, \lambda).C + ((\mathsf{a}, \infty).B \parallel \mathsf{a} \parallel (\mathsf{a}, \lambda).C \\
&\quad = (\mathsf{a}, \lambda).(B \parallel \mathsf{a} \parallel C) + (\mathsf{a}, \lambda).(B \parallel \mathsf{a} \parallel C) \\
&\quad = (\mathsf{a}, 2\lambda).(B \parallel \mathsf{a} \parallel C)
\end{aligned}$$

The solution of this problem in the original definition of EMPA was defective; its revision in a more recent definition [6] essentially boils down to the imposition of certain syntactical requirements to avoid such situations.

IMC. Because of its separation between actions and delays IMC essentially manages to avoid the complications with synchronisation of the other calculi. Synchronising actions does not involve the synchronisation of delays, and delay prefixes do not synchronise, but interleave. Consequently, a (rate) synchronisation function '$*$' is not needed.

By itself, however, this does not guarantee that the IMC approach provides a natural model for synchronisation. To see that this is indeed the case, we 'translate' combined prefixes like $(\mathsf{a}, \lambda).B$ into their IMC counterparts of the form $(\lambda).\mathsf{a}.B$. If we now look at the induced form of (17) under this translation we get:

$$(\lambda).\mathsf{a}.B \parallel \mathsf{a} \parallel (\mu).\mathsf{a}.C = (\lambda).(\mu).\mathsf{a}.(B \parallel \mathsf{a} \parallel C) + (\mu).(\lambda).\mathsf{a}.(B \parallel \mathsf{a} \parallel C) \qquad (29)$$

The right-hand side indicates that action a will take place after a delay of $(\lambda).(\mu)$ or $(\mu).(\lambda)$, whichever is fastest. This is equivalent to a delay with the distribution of the stochastic value that is the maximum of the two exponential delays. This has a very natural operational interpretation: when synchronising the delay is determined by the slowest synchronisation party. The Markovian process algebras that combine actions and delays cannot handle this situation, because the maximum of two exponential distributions is no longer an exponential distribution itself, and therefore falls outside the scope of the model. As in IMC delays can be represented by combinations of exponential delay transitions, it can accommodate such non-exponential distributions within its model. It can, in fact, represent delays from the much larger class of *phase-type distributions* [43], which can approximate general continuous distributions arbitrarily closely (i.e. it is a *dense* subset of the set of continuous distributions).

6 Conclusion

In this paper we have shown how continuous time Markov chain models can be integrated in the process algebraic framework for the modelling and analysis of reactive systems. To do so, we have reviewed the main ingredients of standard process algebra and introduced the basic concepts of continuous time Markov chains. We have observed how Markov chains can be interpreted as transition systems that can be described by process algebraic means, yielding an algebra of Markov chains.

The proces algebraic treatment of Markov chains immediately induces a compositional framework for their representationa and analysis. Syntactically, (large) Markov chains can be represented as the concurrent composition of simpler chains. Semantically, the stochastic version of strong bisimulation equivalence captures exactly the lumpability criterion for Markov chains that is used to simplify chains by the aggregation of equivalent states. As strong bisimilarity is a congruence relation w.r.t. the process algebraic operators, such simplifications can be carried out componentwise (or compositionally) in the algebraic framework, which greatly improves the practical applicability of the method for large chains.

As a next step we have shown that the algebra of Markov chains itself can be merged successfully with a standard process algebra over actions. In particular we have presented the algebra of Interactive Markov Chains (IMC), which can be used to model systems that have two different types of transitions: Markovian delays (represented by their rates), and actions (represented by their action names). We have shown that in IMC we can define both a strong and a weak variant of stochastic bisimulation. Just like in the standard theory weak bisimilarity is not a congruence w.r.t. the choice operator, but a suitable weak congruence can be identified in the canonical way.

IMC provides a process algebraic framework for the integrated modelling and analysis of both functional and (Markovian) performance aspects of reactive systems. Markov chain models can be obtained from the integrated models by abstraction of all observable system actions and subsequent simplification modulo weak bisimulation. The latter can be done compositionally by applying reduction modulo weak congruence componentwise. Of course, this may involve the resolution of remaining nondeterminism.

In the last part of our survey we have compared IMC with a number of other (Markovian) stochastic process algebra's that have been developed with similar goals. In contrast to IMC the other approaches do not have a separation between action transitions and delays, but instead combine them into composite actions of the form (a, λ), meaning that action a can occur only after an exponentially distributed delay with rate λ. In many respects these algebras are quite comparable to IMC. The main exception is the treatment of action synchronisation, which in IMC is straightforward and follows standard process algebra. The other approaches differ according to the different mechanisms by which the rates of the synchronised actions are determined. In IMC delays are not synchronised but interleaved, which for exponential distributions is equivalent to waiting for

the longest delay. This seems an intuitively natural choice. Because of the separation between delays and actions the treatment of synchronisation in IMC is quite elegant, and in our opinion the preferred approach when the maximal delay interpretation of synchronisation applies.

Acknowledgements. Pedro R. D'Argenio, Ulrich Herzog, Rom Langerak, Joost-Pieter Katoen, Lennard Kerber, Michael Rettelbach, Markus Siegle, and Vassilis Mertsiotakis are all kindly acknowledged for their support, ideas and cooperation in our joint research of stochastic process algebra over the past years. The second author is supported by the Netherlands Organisation of Scientific Research (NWO).

References

1. Jos Baeten and Peter Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Computer Science*. Cambridge University Press, 1990.
2. G. Balbo. Introduction to Stochastic Petri Nets. This volume.
3. J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science Publishers, 2001.
4. M. Bernardo and R. Gorrieri. Extended Markovian Process Algebra. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory (7th International Conference, Pisa, Italy, August 1996)*, volume 1119 of *Lecture Notes in Computer Science*. Springer, 1996.
5. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14, 1987.
6. M. Bravetti and M. Bernardo. Compositional asymmetric cooperations for process algebras with probabilities, priorities, and time. In *Proc. of the 1st International Workshop on Models for Time Critical Systems*, volume 39 (3) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2000.
7. M. Bravetti and R. Gorrieri. A complete axiomatisation for observational congruence of prioritized finite state behaviours. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *Automata, Languages, and Programming (ICALP)*, volume 1853 of *Lecture Notes in Computer Science*, pages 744–755, Geneva, Switzerland, 2000. Springer.
8. E. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. A stochastic causality-based process algebra. In S. Gilmore and J. Hillston, editors, *Proc. of the 3rd Workshop on Process Algebras and Performance Modelling*. Special Issue of "The Computer Journal", 38(7) 1995.
9. E. Brinksma, A. Rensink, and W. Vogler. Fair Testing. In Insup Lee and Scott Smolka, editors, *Proceedings of 6th International Conference on Concurrency Theory (CONCUR '95, Philadelphia)*, volume 962 of *Lecture Notes in Computer Science*. Springer, 1995.
10. S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(3):560–599, 1984.
11. P. Buchholz. Markovian Process Algebra: Composition and Equivalence. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, Regensberg/Erlangen, July 1994. Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg.

12. R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clock. In A. Mazurkievicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory (8th International Conference, Warszaw, Poland, August 1997)*, volume 1243 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 1996.
13. P.R. D'Argenio and C. Verhoef. A conservative extension theorem in process algebras with inequalities. *Theoretical Computer Science*, 177:351–380, 1997.
14. P. Darondeau. An Enlarged Definition and Complete Axiomatisation of Observational Congruence of Finite Processes. In M. Dezani-Ciancaglini and U. Montanari, editors, *International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 1982.
15. S. Donatelli. Superposed Generalised Stochastic Petri Nets: Definition and Efficient Solution. In M. Silva, editor, *Proc. of 15th Int. Conference on Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 258–277. Springer, 1994.
16. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic Process Algebras. In *Performance'93*, 1993.
17. J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118:263–299, 1993.
18. B. Haverkort. Markovian Models for Performance and Dependability Evaluation. This volume.
19. H. Hermanns. *Interactive Markov Chains*. PhD thesis, Universität Erlangen-Nürnberg, September 1998. Arbeitsberichte des IMMD 32/7.
20. H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 2001. to appear.
21. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras - Between LOTOS and Markov Chains. *Computer Networks and ISDN Systems*, 30(9-10):901–924, 1998.
22. H. Hermanns and J.-P. Katoen. Automated compositional markov chain generation for a plain-old telephone system. *Science of Computer Programming*, 36(1):97–127, 2000.
23. H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, Erlangen-Regensberg, July 1994. IMMD, Universität Erlangen-Nürnberg.
24. H. Hermanns and M. Siegle. Bisimulation Algorithms for Stochastic Process Algebras and their BDD-based Implementation. In J.-P. Katoen, editor, *ARTS'99, 5th Int. AMAST Workshop on Real-Time and Probabilistic Systems*, pages 144–264. Springer, LNCS 1601, 1999.
25. U. Herzog. Formal methods for performance evaluation. This volume.
26. J. Hillston. Exploiting structure in solution: Decomposing compositional models. This volume.
27. J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.
28. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985.
29. K. Honda and M. Tokoro. On Asynchronous Communication Semantics. In M. Tokoro, O. Nierstrasz, and P. Wegner, editors, *Object-Based Concurrent Computing 1991*, volume 612 of *Lecture Notes in Computer Science*, pages 21–51. Springer, 1992.

30. ISO. *LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1989.
31. B. Jacobs and J. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *EATCS Bulletin*, 62:222–259, June 1997.
32. J.-P. Katoen. *Quantitative and Qualitative Extensions of Event Structures.* PhD thesis, Centre for Telematics and Information Technology, Enschede, 1996.
33. J.-P. Katoen and P.R. D'Argenio. General distributions in process algebra. This volume.
34. J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Springer, 1976.
35. R. Langerak. A testing theory for lotos using deadlock detection. In E. Brinksma, G. Scollo, and C. A. Vissers, editors, *Protocol Specification Testing and Verification IX*, pages 87–98. North-Holland, 1989.
36. R. Langerak. *Transformations and Semantics for LOTOS.* PhD thesis, University of Twente, 1992.
37. R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:269–310, 1983.
38. R. Milner. A Complete Inference System for a Class of Regular Behaviours. *Journal of Computer and System Science*, 28:439–466, 1984.
39. R. Milner. Process constructors and interpretations. In *Proc. IFIP-WG Information Processing.* North-Holland, 1986.
40. R. Milner. *Communication and Concurrency.* Prentice Hall, London, 1989.
41. U. Montanari and V. Sassone. Dynamic Congruence vs. Progressing Bisimulation for CCS. *Fundamenta Informaticae*, XVI(2):171–199, 1992.
42. V. Natarjan and R. Cleaveland. Divergence and Fair Testing. In *ICALP 95*, volume 944 of *Lecture Notes in Computer Science*, pages 648–659. Springer, 1995.
43. M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models–An Algorithmic Approach.* The Johns Hopkins University Press, 1981.
44. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
45. X. Nicollin and J. Sifakis. An Overview and Synthesis on Timed Process Algebras. In J. W. de Bakker, K. Huizing, and W.-P. de Roever, editors, *Real-Time: Theory in Practice (REX Workshop)*, volume 600 of *Lecture Notes in Computer Science.* Springer, 1990.
46. E.-R. Olderog and C.A.R. Hoare. Specification Oriented Semantics for Communicating Processes. *Acta Informatica*, 23:9–66, 1986.
47. D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Fifth GI Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science.* Springer, 1981.
48. Joachim Parrow and Peter Sjödin. The Complete Axiomatization of cs-Congruence. In P. Enjalbert, E. W. Mayr, and K. W. Wagner, editors, *STACS '94*, volume 775 of *Lecture Notes in Computer Science*, pages 557–568. Springer, 1994.
49. I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(2):241–284, 1987.
50. B. Plateau and K. Atif. Stochastic Automata Network for Modeling Parallel Systems. *IEEE Transactions on Software Engineering*, 17(10), 1991.
51. G.D. Plotkin. A Structured Approach to Operational Semantics. Technical Report DAIMI FM-19, Computer Science Department, Aarhus University, 1981.
52. M. Hennessy und T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.
53. A. Valmari and M. Tienari. Compositional Failure-based Semantic Models for Basic LOTOS. *Formal Aspects of Computing*, 7:440–468, 1995.

54. R. J. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
55. R.J. van Glabbeek. The Linear Time – Branching Time Spectrum II: The Semantics of Sequential Systems With Silent Moves (Extended Abstract). In Eike Best, editor, *Fourth International Conference on Concurrency Theory (CONCUR '93, Hildesheim, Germany)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.
56. R.J. van Glabbeek. The Linear Time – Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes. In Bergstra et al. [3], pages 3–99.
57. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.
58. C.A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma. Specification Styles in Distributed Systems Design and Verification. *Theoretical Computer Science*, 89(1):179–206, 1991.
59. W. Yi. CCS + Time = An Interleaving Model for Real Time Systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Eighteenth Colloquium on Automata, Languages and Programming (ICALP) (Madrid, Spain)*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 1991.

Verification of Randomized Distributed Algorithms

Roberto Segala

Department of Computer Science
University of Bologna

Abstract. We describe modular verification techniques for randomized distributed algorithms as extensions of techniques for ordinary, non randomized, distributed algorithms. The main difficulty to overcome arises from the subtle interplay between probability and nondeterminism, where probability is due to the random choices that occur within an algorithm, and nondeterminism is due to the unknown speeds and scheduling policies of the processes. The techniques that we introduce are based on separation of probability from nondeterminism.
When the nondeterminism is factored out, the analysis of an algorithm has several pieces that are in common with the area of performance evaluation. Thus, the techniques that we describe are likely to constitute a bridge to export typical performance evaluation techniques to the area of concurrent nondeterministic systems and, vice versa, to understand alternative ways for handling nondeterminism when it arises.

1 Introduction

Since the pioneering paper of Rabin [81], randomization has turned out to be a fundamental tool for the efficient solution of problems, and in particular, in the field of distributed computation, a fundamental tool for the solution of problems that are otherwise unsolvable. Examples are algorithms for mutual exclusion with few shared variables [82,57], algorithms for consensus [21,12,13], shown otherwise unsolvable in [38], and algorithms for symmetric solution to the dining philosophers problem [65].

Unfortunately, randomization is very complicated to deal with since it is combined with the intrinsic nondeterminism of concurrent systems that derives from the unknown scheduling policies and relative speeds of the processes. As opposed to the standard approach in performance analysis, we cannot simply assume some approximate form of resolution of the nondeterminism; rather we are interested in the worst case scenario. Thus, our main interest is to study the functional correctness and performance of an algorithm under any resolution of the nondeterminism, possibly with some fairness restrictions.

The main difficulty in the analysis of a randomized distributed algorithm is that, although the designer of an algorithm may believe that a specific stochastic process takes place during execution, the resolution of the nondeterminism may affect such stochastic process at the point of compromising its outcome. Several

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 232–260, 2001.

times the resolutor of the nondeterminism, which is usually called an *adversary*, has the ability not to schedule some of the elementary experiments that are part of a stochastic process, thus creating dependencies between events that are supposed to be independent. The result is that several algorithms proposed in the literature turned out to be incorrect. As an example, the original algorithm for mutual exclusion of Rabin [82] was shown to have problems in [84] and later fixed in [57].

Our objective is to study techniques for the analysis of randomized distributed algorithms, possibly extending known techniques for the analysis of ordinary, non randomized, distributed systems. In particular, we extend labeled transition systems [56] to account for randomization, and we extend the typical modular verification techniques based on analysis of computations [68] and simulation relations [71,51,69]. Modular analysis is important for scalability issues, so that the analysis of a large system can be decomposed into the analysis of several smaller systems.

The probabilistic extension of labeled transition systems that we obtain, which we call *probabilistic automata*, turn out to be an extension of Markov Decision Processes [32], a model used for the study of optimal control *policies* in a randomized environment; furthermore, once the nondeterminism is resolved, the objects that we obtain are infinite state Markov Processes, or more precisely semi-Markov Processes, and these semi-Markov processes, which we call *probabilistic executions*, are the objects on which functional correctness and performance is studied.

Large probabilistic automata are built by composing smaller probabilistic automata and letting them interact together by means of a mechanism that resembles the product of ordinary automata. An important result (cf. Theorem 1) is that each probabilistic execution of the composition of two probabilistic automata can be seen as the result of two probabilistic executions of the two components, respectively. This means that a property of the composition of two probabilistic automata can be derived from the properties of the components, i.e., it is possible to reason in a modular way.

We introduce two abstract verification techniques based on the extension of the idea of language inclusion [49], also called *trace inclusion*, and of simulation relation [51,69]. Both techniques induce a preorder relation on probabilistic automata that is preserved by composition, thus leading again to modularity. Roughly speaking, the language of a probabilistic automaton describes all the possible communication patterns that the automaton can engage in. Thus, if the language of a probabilistic automaton $\mathcal{A}$ is a subset of the language of another probabilistic automaton $\mathcal{B}$, then $\mathcal{A}$ cannot engage in any communication pattern that $\mathcal{B}$ cannot engage in, and because of compositionality, $\mathcal{A}$ cannot engage in any communication pattern that $\mathcal{B}$ cannot engage in independently of the context in which $\mathcal{A}$ and $\mathcal{B}$ are embedded. Suppose now that $\mathcal{B}$ always responds to the requests it receives. Then there are no traces in the language of $\mathcal{B}$ where a request does not have any response. Because of trace inclusion, there are no traces in the language of $\mathcal{A}$ where a request does not have any response, and

thus $\mathcal{A}$ responds to all the requests it receives. Language inclusion can be used as an implementation relation. Simulation relations are used as a sound proof technique for language inclusion.

We also describe two techniques for the separation of probability from nondeterminism, since it is the interaction between probability and nondeterminism that causes most of the problems. The first technique consists of a collection of tools, called *coin lemmas*, that provide us with a rule to associate an event with each probabilistic execution of a probabilistic automaton and a minimum probability p that each event is guaranteed to have. Thus it is sufficient to show that all the elements in the events given by the rule ensure the correct termination of the algorithm under examination to conclude that the algorithm terminates correctly with probability at least p. The advantage of coin lemmas is that they take care explicitly of the interaction between probability and nondeterminism (cf. Sections 3 and 5), and reduce the analysis of a probabilistic property to the analysis of a property that involves only nondeterminism.

The second technique for the separation of probability and nondeterminism originates from arguments that are typical of random variables. Consider a probabilistic execution (a semi-Markov process), and consider a function ϕ that assigns a non-negative real number with each path (which we call an execution) in the probabilistic execution. We call ϕ a *complexity function*. Consider a reachability property that holds with probability 1. Function ϕ applied to the minimum paths where the reachability property is satisfied is a random variable that expresses the complexity required to satisfy the reachability property. Furthermore, the expected value of ϕ is the expected complexity to satisfy the property. In distributed algorithms it is typical to study the relationship between different complexity measures to derive the complexity of the algorithm under examination. Such relations can be lifted to expectations using the results that hold for random variables. Once again, results about expectations are proved by analyzing properties that involve only nondeterminism.

Throughout the paper we give highlights of a large case study [79] where all the techniques that we present are used. We do not work out all the details of the example and we refer the interested reader to the paper that documents the whole case study. Other simpler case studies appear in [2,67,78,86].

The fact that in our analysis we have to deal with objects that are similar to Markov Decision Processes and with Markov processes in general suggests that the analysis of randomized distributed algorithms has several elements in common with classical performance analysis. It is our hope that the experience we have gained can constitute one of the bridges for exchanging techniques between concurrency theory, specifically the study of nondeterminism in concurrency theory, and performance analysis. Our presentation is focused on algorithms; yet people familiar with other problems that include randomization and that need an explicit treatment of nondeterminism may find useful suggestions.

The rest of the paper is structured as follows: Section 2 gives an overview of related work in the areas of probabilistic models with nondeterminism and of analysis of randomized distributed algorithms; Section 3 illustrates some ex-

amples of typical problems that arise in the analysis of randomized distributed algorithms; Section 4 introduces probabilistic automata and the related modular verification techniques; Section 5 introduces coin lemmas; Section 6 shows how to apply the techniques of this paper by giving highlights on a specific example that is studied in more detail in [79]; Section 7 studies techniques for performance analysis, and Section 8 shows how to analyze the performance of the algorithm of Section 6; finally, Section 9 gives some concluding remarks.

2 Related Work

The study of probability in concurrency theory dates back to the 80's [62], where the nondeterministic + operator was replaced by a probabilistic + operator. Since then, several models were proposed, and a first classification attempt came in [41,40], where probabilistic processes were classified into reactive, generative, and stratified. Typical reactive models are in [62,24,63,64,27] and typical generative models are in [39,54,92,14,31,11]. A generative process is like a Markov process and does not include any nondeterminism, while a reactive process includes some form of nondeterminism. A mixed model was introduced in [94,95]. Denotational models for probabilistic systems were studied in [50,91]. Models with real nondeterminism and probability were introduced in [45,46,96,89].

Several concepts from concurrency theory are extended to the probabilistic case. Among these are simulation relations [52,89], trace based semantics [39,86], and testing based semantics [26,96,27,73,87]. In the area of probabilistic bisimulation there is a lot of interest in decision algorithms [18,75,20] and in variants of bisimulation that lead to more efficient algorithms [20].

Other studies that combine concurrency theory and performance analysis are stochastic process algebras [48,42,22]. The reader interested in this area may find more specific information in this volume [25,47]. Typically stochastic process algebras include little nondeterminism and are more alike generative processes.

Nondeterminism was studied also in the area of model checking [36], where the validity of a formula expressed in some temporal logic is verified mechanically on a model expressed in terms of automata. The pioneering paper was in 1985 [93], where a simple model checking algorithm for *Concurrent Markov Chains* was introduced. Concurrent Markov Chains are very much alike probabilistic automata if we do not dive too deeply into semantic issues. Other model checking algorithms for properties that hold with probability 0 or 1 are studied in [29,30], and extensions with real time are studied in [9,10]. The first model checking algorithm for properties that hold with any probability is presented in [43,44], and more efficient algorithms are studied in [23]. Fairness is studied in [19], and symbolic model checking algorithms are studied in [17,8]. Quantitative properties in the presence of real time are studied in [59,60].

Other relevant work in the analysis of systems with probability and nondeterminism is in [4,6,5,7]. Here, some automatic techniques for the analysis of long average behavior properties of systems are introduced. The models and techniques are inspired by former studies on Markov Decision Processes [32], a

model that in essence coincides with reactive processes and that was introduced to study optimal control policies for systems that include randomization.

Finally, objects like probabilistic automata were introduced long ago in [80] for the study of generalizations of the class of regular languages.

In the area of verification of randomized distributed algorithms there is work on extending temporal logic based techniques in [76,77,83,70]. These techniques can be used to study events that hold either with probability 1 or probability 0 and are based on transforming the 0/1 probability of an event into a topological property of the underlying transition system. The main underlying concepts are called extreme fairness and α-fairness. The same techniques were used for the study of performance [76], but the results are not very tight.

Other logic-based techniques are introduced in [70], where the reasoning style of probabilistic predicate transformers [72] is applied to the analysis of a distributed algorithm. Predicate transformers are a generalization of the weakest precondition style of reasoning; their semantics is a real number rather than a boolean, and the real number is an expectation. Thus, the performance of an algorithm is the semantics of the formula that expresses its complexity.

In recent work [35] a new idea of scheduler-luck games was introduced. Given a randomized distributed algorithm we set up a game between a player *scheduler* and a player *luck*. The player scheduler resolves the nondeterminism trying to compromise the correctness of the algorithm, while the player luck fixes the outcome of some of the coin flips trying to ensure the correctness of the algorithm. We say that luck has a k winning strategy if it can ensure correctness against any scheduler by fixing the value of at most k coins. In such case we can conclude that the algorithm is correct with probability at least $1/2^k$. Scheduler-luck games can be seen as an instance of the coin lemmas that we introduce in Section 5; however in some cases they provide us with a very elegant verification technique.

3 The Problems with Randomization

We can view a distributed system as a collection of tasks that evolve on separate processors and that communicate through messages. A distributed algorithm is a description of the tasks that each processor should perform, and a distributed algorithm is called randomized if some of the operations performed by the processors involve some randomization. The evolution of a randomized distributed system can be described as a stochastic process; however, the stochastic process changes depending on the relative speeds of the processors and/or the scheduling policy between the processors. Thus, since processor speeds and scheduling policies are generally unknown, there is no way to know exactly the stochastic process that will take place. Stating that an algorithm is correct means stating that no matter what stochastic process takes place the algorithm achieves its goal.

Showing the correctness of a randomized distributed algorithm is not easy at all. To cite some authoritative comments, "intuition often fails to grasp the full intricacy of the algorithm" [76], and "proofs of correctness for probabilistic

distributed systems are extremely slippery" [65]. Moreover, the difficulty of the problem is supported by the existence of are several flawed algorithms.

One problem in the analysis of an algorithm is that it is difficult to define precisely the probability space on which to reason, and the lack of a well defined probability space forces the designer of an algorithm to reason informally. This is the problem at the base of the error in the analysis of the the first randomized algorithm of Rabin for mutual exclusion [82]: one of the informal steps in the proof relates two probabilities that are not defined in the same probability space.

Another problem derives from the fact that sometimes the stochastic process that a designer anticipates may not take place in the algorithm. We illustrate the problem by referring to a case study [2]. In [3] a randomized algorithm for computing a spanning tree in a network is presented. In the algorithm each node tries to detect what tree to belong to and in particular each node decides whether it is a root of its tree by using randomization to break symmetry. Roughly, when two or more nodes are in conflict because they all want to be a root of the same tree, the nodes throw a random number, called an id, according to the Afek-Matias schema [1], i.e., number i is chosen with probability $1/2^i$. Then the root is the process that draws the highest id. It is shown in [1] that there is a constant c such that, whenever k processes draw a number according to the Afek-Matias distribution (k is any fixed number), with probability at least c there is a unique maximum. Then we can conclude that there is a unique root with probability at least c, which is sufficient to show the correctness of the algorithm.

Unfortunately, a closer look at the argument above shows that there is a serious flaw. Specifically, the result of [1] is stated in terms of conditional probabilities, in the sense that if we first fix k and then we draw k numbers, then there is a unique maximum with probability at least c. However, if we choose how many id's to draw based on some partial outcome of the process, then we could continue to draw id's until we have two maximums. In such case the probability of a unique maximum would be very low. Thus, in order to show the correctness of the algorithm of [3], we need to make sure that it is not possible to schedule processes so that the number of id's to draw can be determined based on some partial outcome of the drawing process. Indeed, the problem was present in a previous unpublished version of the algorithm and it was discovered only when one of the authors tried to apply coin lemmas to verify its correctness. We will return to this problem when talking about coin lemmas.

4 Probabilistic Automata

In this section we describe probabilistic automata [86], the model that we use for our analysis. All our definitions are given in the discrete case; generalizations to non-discrete systems appear in [34,33].

4.1 Probabilistic Automata

A *probabilistic automaton* $\mathcal{A}$ is a tuple $(S, \bar{s}, \Sigma, \mathcal{D})$ where S is a set of *states*, $\bar{s} \in S$ is a *start state*, $\Sigma = (E, H)$ is a pair of disjoint sets of *external* and *internal*

(hidden) actions, respectively, called an *interface*, and $\mathcal{D} \subseteq S \times \Sigma \times Disc(S)$ is a *transition relation*, where $Disc(S)$ denotes the set of discrete probability measures over S.

States are ranged over by s, q; actions are ranged over by a, b, c; internal actions are ranged over by τ; discrete distributions are ranged over by μ. We call an element of $\mathcal{D}$ a *transition*, and we denote a generic transition (s, a, μ) alternatively by $s \xrightarrow{a} \mu$. We say that a transition (s, a, μ) is *enabled* from s and that it is *labeled* by a. We call a transition (s, a, μ) *Dirac* if μ is a Dirac measure, i.e., a measure that assigns probability 1 to a single element. We call a state s *Dirac* if all the transitions enabled from s are Dirac; we call s *deterministic* if it enables at most one transition for each action; we call s *single* if it enables at most one transition. We say that a probabilistic automaton is *deterministic* if each state is deterministic.

For notational convenience, we denote the elements of a generic probabilistic automaton $\mathcal{A}$ by $S, \bar{s}, \Sigma, E, H, \mathcal{D}$, and we propagate primes and indicies. Thus, the elements of $\mathcal{A}'_i$ are denoted by $S'_i, \bar{s}'_i, \Sigma'_i, E'_i, H'_i, \mathcal{D}'_i$. Sometimes we abuse of notation and write Σ for the set $E \cup H$ rather than the pair (E, H). Finally, for a state s we denote by $\mathcal{D}(s)$ the set of transitions of $\mathcal{D}$ that are enabled from s.

If we think of a Markov process, for each state there is a unique probability distribution over states that describes the probability distribution of the next state in the stochastic process. In probabilistic automata the distribution is not unique; rather there are several distributions. From the formal point of view, in a Markov process $\mathcal{D}$ is a function rather than a relation. In the description of a distributed algorithm nondeterminism arises from the fact that from each state several processes are ready to perform a computational step, but only one of them will be the first one to take its step. Each computational step is described by a transition; thus, for each state there are several transitions enabled. Probabilistic automata are equipped also with a set of actions that take an active role in the transitions. Actions are used to describe potential communication between automata when they interact (cf. Section 4.3).

An ordinary labeled transition system (automaton) [56] is essentially a probabilistic automaton where all states are Dirac. The probabilistic automata of [86] are more general than the probabilistic automata defined here in that $\mathcal{D} \subseteq S \times Disc(\Sigma \times S \cup \bot)$, where $\bot$ is a special symbol that denotes the possibility of not moving from s forever. The probabilistic automata of this paper coincide with the simple probabilistic automata of [86]. The reactive systems of [62] are deterministic probabilistic automata. The probabilistic automata of [80] are essentially our probabilistic automata. Markov Decision Processes [32] are essentially deterministic probabilistic automata. The *alternating model* [93] can be seen as a probabilistic automaton where each state is either Dirac or single.

4.2 Probabilistic Executions

An *execution* of a probabilistic automaton $\mathcal{A}$ is an alternate sequence of states and actions, starting with a state and, if the sequence is finite, ending with a state, $\alpha = s_0 a_1 s_2, \ldots$, such that for each i there exists a transition (s_i, a_{i+1}, μ) in

$\mathcal{D}$ such that $\mu(s_{i+1}) > 0$. An execution describes one of the possible evolutions of a probabilistic automaton. It is the result of resolving the nondeterminism and the probability as well.

Denote by $fstate(\alpha)$ the first state of an execution α and, if α is finite, denote by $lstate(\alpha)$ the last state of α. A finite execution $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of A and an execution $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of A can be *concatenated*. The concatenation, written $\alpha_1 \frown \alpha_2$, is the execution $s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$. An execution α_1 of $\mathcal{A}$ is a *prefix* of an execution α_2 of $\mathcal{A}$, written $\alpha_1 \leq \alpha_2$, iff either $\alpha_1 = \alpha_2$ or α_1 is finite and there exists an execution α_1' of $\mathcal{A}$ such that $\alpha_2 = \alpha_1 \frown \alpha_1'$.

To study the probabilistic behavior of a probabilistic automaton we need to study the objects that result from the resolution of the nondeterminism only, which are semi-Markov processes. We define a function σ, called a *scheduler*, that resolves the nondeterminism in a probabilistic automaton based on the past history. That is, σ is a function that takes a finite execution and returns a sub-probability distribution over the transitions that are enabled from the last state of its argument. Formally, $\sigma(\alpha) \in SubDisc(\mathcal{D}(lstate(\alpha)))$. If in $\sigma(\alpha)$ the measure of $\mathcal{D}(lstate(\alpha))$ is not 1, then with probability $1 - \sigma(\alpha)(\mathcal{D}(lstate(\alpha)))$ no transition is scheduled. In the area of distributed algorithms schedulers are usually called *adversaries*, since they are seen as entities that try to degrade the performance of an algorithm as much as possible, while in the area of Markov Decision Processes schedulers are called *policies*, since the main objective is to find the scheduler (controller) that achieves the best performance.

If we fix a starting state s and a scheduler σ, then we have determined a semi-Markov process with start state s. Indeed, for each finite execution α that starts with s, $\sigma(\alpha)$ determines the distribution over the next action and state in the process. We can then compute the probability of each finite execution and define a probability distribution over executions using the classical cone construction. Specifically, we need to build a collection of sets of executions, called a σ-field, that includes the empty set and is closed under complement and countable union. The elements of the σ-field are the objects that we can measure. According to the cone construction, a cone is a set of the form $C_\alpha = \{\alpha' \mid \alpha \leq \alpha'\}$, where α is a finite execution that starts with s. We consider the σ-field generated by the set of cones, that is, the smallest σ-field that includes all the cones, and we define the probability of a cone C_α as the probability of α. Standard measure theoretic arguments show that the measure defined on the cones extends uniquely to a measure defined over the whole σ-field. We denote such measure by $\mu_{\sigma,s}$.

Observe that a cone C_α contains exactly all those executions where α occurs, and thus it describes the occurrence of α. For this reason it is reasonable to define $\mu_{\sigma,s}(C_\alpha)$ as the probability of α. To be convinced that the σ-field generated by the cones is sufficiently expressive, simply observe that any reachability property (e.g., the occurrence of some action a or the reachability of some state s) can be expressed as a union of cones (the cones of execution where the reachability property is successful, e.g., a occurs or s is reached) and that the set of cones with non-zero probability is countable.

At this point the typical statements in the analysis of an algorithm are of the form "for each probabilistic execution the probability of an event E is at least p". Of course the event E must be a measurable set under any scheduler. An event E could be the set of execution where each request has a response.

4.3 Parallel Composition

We said in several occasions that modular reasoning is essential. That is, we said that we want to analyze the properties of a large system by analyzing its components separately. Parallel composition is the mechanism that allows us to compose small components to lead to a large system. In this section we introduce the parallel composition operator and its main property that enables modular reasoning.

Parallel composition is also the operator that illustrates how two or more probabilistic automata communicate. Each probabilistic automaton has an interface, which is the set of communications that the probabilistic automaton can engage in. Following the approach of CSP [49], if an action is in common between two probabilistic automata, then we force synchronization on such action: in the composition a probabilistic automaton performs the common action iff also the other probabilistic automaton performs the same action. In such case a communication takes place. The reader interested in action-based communication is referred to the extensive literature on proces algebras [49,71,16].

We now turn to the formal definitions. We say that two probabilistic automata $\mathcal{A}_0$ and $\mathcal{A}_1$ are *compatible* is $E_0 \cap H_1 = H_0 \cap E_1 = \emptyset$. The parallel composition of two compatible probabilistic automata $\mathcal{A}_0, \mathcal{A}_1$, denoted by $\mathcal{A}_0 \| \mathcal{A}_1$, is a probabilistic automaton $\mathcal{A}$ defined as follows:

- $S = S_0 \times S_1$;
- $\bar{s} = (\bar{s}_0, \bar{s}_1)$;
- $E = E_0 \cup E_1$ and $H = H_0 \cup H_1$;
- $\mathcal{D}$ is the set of triplets $((s_0, s_1), a, \mu_1 \times \mu_2)$, where, for $i \in \{0, 1\}$, $s_i \in S_i$, and either
  - $(s_i, a, \mu_i) \in \mathcal{D}_i$, or
  - $a \notin \Sigma_i$ and $\mu_i(s_i) = 1$.

The definition of $\mathcal{D}$ needs some more words of explanation. The measure $\mu_0 \times \mu_1$ considers the experiments described by μ_0 and μ_1 as independent; thus, $\mu_0 \times \mu_1(s_0, s_1)$ is defined to be $\mu_0(s_0)\mu_1(s_1)$. The automata $\mathcal{A}_0$ and $\mathcal{A}_1$ synchronize on their common actions and evolve independently on the other actions. The second item in the definition of $\mathcal{D}$ states that an automaton $\mathcal{A}_i$ idles whenever $\mathcal{A}$ performs a transition labeled by an action not in Σ_i. In other words, $\mathcal{A}_{1-i}$ evolves independently with such action. Observe that from the compatibility condition two probabilistic automata cannot engage in synchronizations through their internal actions.

The main mechanism to derive properties of a probabilistic automaton from the properties of its components is the concept of *projection*. That is, we can

define the projections of a probabilistic execution onto the component automata and prove that each projection is indeed a probabilistic execution of the component. Later in Section 6.4 we can see this result at work.

Let $\mathcal{A}$ be $\mathcal{A}_0 \| \mathcal{A}_1$, and let α be an execution of $\mathcal{A}$. For $i \in \{0, 1\}$, define $\pi_i(\alpha)$, the projection of α onto $\mathcal{A}_i$, to be the sequence obtained from α by projecting each state onto its i^{th} component and by removing all the actions not in Σ_i together with their following state. It is immediate to show that $\pi_i(\alpha)$ is an execution of $\mathcal{A}_i$.

If we apply the projection operator to a probabilistic execution $\mu_{\sigma,s}$, then it is possible to show that the projection is a measurable function from the σ-field relative to s and the σ-field relative to $\pi_i(s)$. Thus, the measure induced by π_i, which we denote by $\pi_i(\mu_{\sigma,s})$, is a well defined probability measure. Recall from probability theory that the induced measure $\pi_i(\mu_{\sigma,s})$ is defined as $\pi_i(\mu_{\sigma,s})[E] = \mu_{\sigma,s}[\pi_i^{-1}(E)]$ The main result is then the following.

Theorem 1. *Let $\mathcal{A}$ be $\mathcal{A}_0 \| \mathcal{A}_1$, and let $\mu_{\sigma,s}$ be a probabilistic execution of $\mathcal{A}$. Let $i \in \{0, 1\}$. Then there is a scheduler σ_i for $\mathcal{A}_i$ such that $\pi_i(\mu_{\sigma,s}) = \mu_{\sigma_i,\pi_i(s)}$.*

4.4 Trace Distributions

The trace of an execution α is the sub-sequence of α that consists of the external actions. That is, a trace extracts from an execution the sequence of externally visible communication events that take place. Several properties can be studied simply by looking at the communication events of a system. As an example, if in a system requests and responses occur through communication events, then we can see whether each request obtains a response by observing traces.

The function that extracts the trace of an execution is a measurable function from the σ-field associated with a probabilistic execution and the σ-field over traces generated by cones of traces. Thus, a probabilistic execution can be associated with its corresponding probability distribution over traces, which we call a *trace distribution*.

Two probabilistic automata can be compared based on their trace distribution sets in a similar way as two ordinary automata are compared based on their languages. We say that $\mathcal{A} \sqsubseteq_D \mathcal{A}'$ if each trace distribution of $\mathcal{A}$ is also a trace distribution of $\mathcal{A}'$. Usually we say that $\mathcal{A}$ *implements* $\mathcal{A}'$.

As an example consider the property stating that a system responds to its requests with probability at least $1/2$. If requests and responses are external actions, then in $\mathcal{A}'$ each request obtains a response with probability at least $1/2$ if there is no trace distribution of $\mathcal{A}'$ where a request obtains a response with probability lower than $1/2$. Since $\mathcal{A} \sqsubseteq_D \mathcal{A}'$, then also in $\mathcal{A}$ there is no trace distribution where a request obtains a response with probability lower than $1/2$. Thus, also in $\mathcal{A}$ each request obtains a response with probability at least $1/2$.

Trace distribution inclusion is not preserved by parallel composition, which is a problem if our objective is to reason in a modular way. Roughly speaking, two probabilistic automata that have the same traces but resolve the nondeterminism in different points (say $\mathcal{A}_0$ resolves the nondeterminism before $\mathcal{A}_1$) can be

distinguished by a probabilistic automaton flips a coin and publishes the result after $\mathcal{A}_0$ resolves the nondeterminism and before $\mathcal{A}_1$ resolves the nondeterminism. In this case $\mathcal{A}_1$ can resolve its nondeterminism based on the outcome of the coin flip of the context, while $\mathcal{A}_0$ cannot. See [85] for some detailed examples. For this reason we introduce a new relation called *trace distribution precongruence*, denoted by $\sqsubseteq_C$, which is the coarsest precongruence included in the trace distribution inclusion.

Studying the trace distribution precongruence directly is complicated; however we can use simulations as illustrated in the next section as a sufficient condition for trace distribution precongruence. This kind of approach is typical in the analysis of distributed systems and will be illustrated later in this paper (cf. Section 6.6).

4.5 Simulation Relations

Proving trace distribution inclusion is difficult in general. However, if we can show that each move of a probabilistic automaton $\mathcal{A}_0$ can be simulated by a probabilistic automaton $\mathcal{A}_1$, then we can conclude that each trace distribution of $\mathcal{A}_0$ is also a trace distribution of $\mathcal{A}_1$. This idea is known as the simulation method [69] and constitutes a very useful technique for the analysis of distributed systems [66]. In this section we describe one of the probabilistic extensions of the notion of simulation and state its properties with respect of modularity and trace distribution inclusion. We refer the interested reader to [90,86].

To define formally the notion of simulation we need to define the notion of *weak transition*. We say that there is a weak transition from a state s labeled by a to distribution μ, denoted by $s \stackrel{a}{\Longrightarrow} \mu$, if there exists a scheduler σ such that in $\mu_{\sigma,s}$ the following holds:

- the support of $\mu_{\sigma,s}$ is included in the set of finite executions that start with s and have trace a;
- for each state s', $\mu(s') = \sum_{\alpha | lstate(\alpha)=s'} \mu_{\sigma,s}(\alpha)$.

In other words there is a weak transition from s labeled by a to μ if it is possible to schedule transitions in such a way that in the resulting semi Markov process we terminate with probability 1 after performing several internal action and a single external action a, and the distribution over the terminal states is μ.

A simulation relation from a probabilistic automaton $\mathcal{A}_0$ to a probabilistic automaton $\mathcal{A}_1$ is a relation $\mathcal{R} \subseteq S_0 \times S_1$ such that

- $\bar{s}_0 \mathcal{R} \bar{s}_1$, and
- if $s_0 \mathcal{R} s_1$ and $s_0 \stackrel{a}{\longrightarrow} \mu_0$, then there exists a distribution μ_1 such that $s_1 \stackrel{a}{\Longrightarrow} \mu_1$ and $\mu_0 \mathcal{R} \mu_1$.

We write $\mathcal{A}_0 \sqsubseteq \mathcal{A}_1$ if there is a simulation from $\mathcal{A}_0$ to $\mathcal{A}_1$.

We have not said yet what it means that $\mu_0 \mathcal{R} \mu_1$. If μ_0 and μ_1 are Dirac distributions, then we can simply say that the states that have probability 1 are related. In general the definition is slightly more complicated since a state in the

support of μ_0 could be related to several states in the support of μ_1 and vice versa. What counts is that each state s_0 in the support of μ_0 is related to states in the support of μ_1 that have a cumulative probability equal to $\mu_0(s_0)$ and vice versa. Formally, $\mu_0 \mathcal{R} \mu_1$ if there is a function $w : S_0 \times S_1 \rightarrow [0, 1]$ such that

- if $w(s_0, s_1) > 0$ then $s_0 \mathcal{R} s_1$;
- for each $s_0 \in S_0$, $\sum_{s_1 \in S_1} w(s_0, s_1) = \mu_0(s_0)$;
- for each $s_1 \in S_1$, $\sum_{s_0 \in S_0} w(s_0, s_1) = \mu_1(s_1)$.

The two main properties of simulation relations are the fact that simulation relations are preserved by parallel composition and that they imply trace distribution inclusion. Combining the two facts, simulation relations imply trace distribution precongruence.

Theorem 2. *Let $\mathcal{A}_0, \mathcal{A}_1$ be two probabilistic automata such that $\mathcal{A}_0 \sqsubseteq \mathcal{A}_1$, and let $\mathcal{A}$ be a probabilistic automaton compatible with both $\mathcal{A}_0$ and $\mathcal{A}_1$. Then,*

- *$\mathcal{A}_0 \sqsubseteq_C \mathcal{A}_1$, and*
- *$\mathcal{A}_0 \| \mathcal{A} \sqsubseteq \mathcal{A}_1 \| \mathcal{A}$.*

5 Coin Lemmas

Stating that a property ϕ holds with some minimum probability p no matter what the adversary does means that for each probabilistic execution the event that expresses ϕ has probability at least p. The property ϕ could state that an algorithm terminates successfully. Unfortunately, it is usually the case that analyzing ϕ directly is very difficult. Thus, rather than identifying the event that expresses ϕ, we can identify a sub-event that is guaranteed to have probability at least p. It is even better and less error prone if we can identify a simple rule to associate an event of probability at least p with each probabilistic execution.

A possible rule consists of identifying a set of executions θ and associate each probabilistic execution μ with the event $\Omega_\mu \cap \theta$, where Ω_μ denotes the sample space associated with μ. With such a rule, once we know that each execution of θ ensures ϕ, we can conclude that ϕ holds with probability at least p, where p is the minimum probability ensured by the rule. The main advantage of this approach is that the analysis of a property that involves probability is reduced to a property that does not involve probability. The problem that we have to solve is how to choose a rule that would guarantee the minimum probability p. Coin lemmas [86,67,88] serve this purpose.

The choice of a rule is not as simple as it might appear. As a simple example, consider an algorithm where at some time two processes may flip one coin each, and suppose that the algorithm terminates successfully whenever the two coins give the same result. Something along these lines occurs in the randomized consensus algorithm of Ben Or [21]. We would like to say that the algorithm terminates successfully with probability at least 1/2. However, there may be executions of the algorithm where either no coin is flipped or exactly one coin is flipped. In order to state that the algorithm is successful with probability

at least 1/2 we have to show either that two coins are always flipped, or that the algorithm terminates successfully whenever less than two coins are flipped. Problems like this one are very easy to overlook. Alternatively, we can formulate a coin lemma stating that, if we define θ to be the set of executions where either less than two coins are flipped or two coins are flipped and they give the same result, then for each probabilistic execution μ the probability of $\Omega_\mu \cap \theta$ is at least 1/2. Then it is sufficient to consider all the executions of θ without worrying about probabilities. Observe that from the definition of θ we are forced to consider explicitly the possibility of flipping less than two coins.

We now give a formal statement of the coin lemma cited above, where we use actions to identify the coins that are flipped.

Lemma 1. *Let $\mathcal{A}$ be a probabilistic automaton, and let $(a_1, U_1), (a_2, U_2)$ be pairs consisting of an action and a set of states such that $a_1 \neq a_2$. Let p_1, p_2 be two numbers in $[0, 1]$ such that, for $i \in \{1, 2\}$, for each transition (s, a_i, μ) of $\mathcal{A}$, $\mu[U_i] \geq p_i$.*

For each probabilistic execution μ let $FIRST((a_i, u_i))(\mu)$ be the set of executions α of Ω_μ such that either a_i does not occur in α, or a_i occurs in α, and its first occurrence leads to a state of U_i.

Then, for each probabilistic execution μ of $\mathcal{A}$,

- $P_\mu[FIRST((a_i, U_i))(\mu)] \geq p_i$.
- $P_\mu[FIRST((a_1, U_1))(\mu) \cap FIRST((a_2, U_2))(\mu)] \geq p_1 p_2$.

The main idea behind the formulation of a coin lemma is the following: we fix a collection of elementary experiments and we fix a collection of successful outcomes. Let p be the probability of the successful outcomes. Then we map each elementary experiment to some random draws in a probabilistic execution in such a way that no two distinct elementary experiment map to the same random draw. If in an execution not all the elementary experiments take place, then the execution is considered by the rule if and only if it is possible to fix the outcome of the elementary experiments that do not take place in such a way that we obtain a successful outcome for the whole collection of elementary experiments. Then we are guaranteed that the chosen set of executions always has probability at least p.

Returning to the algorithm of [3] discussed in Section 3, the coin lemma for the experiment of throwing k id's would consider the set θ of executions where either less than k id's are drawn, or exactly k id's are drawn and there is a unique maximum. Then the probability of θ is at least c, where c is the probability identified in Section 3. A scheduler that draws new id's until there are two maximums would stop sometimes after drawing less than k id's, and in the corresponding execution, which is considered in the coin lemma since we can fix the values of the unflipped id's so that there is a unique maximum, we would discover that there is more than one winner, thus revealing the flaw in the algorithm.

The reader interested in the analysis of the algorithm of [3] is referred to [2]; the reader interested in the analysis of the algorithm of Ben Or is referred to [86]; the reader interested in details about coin lemmas is referred to [86,88].

6 An Example: Randomized Consensus

In this section we give highlights of a case study [79] where the randomized consensus algorithm of Aspnes and Herlihy [12] is proved to be correct and terminate within expected cubic time. We do not work out the analysis in full detail; rather we describe the parts of the proof that show the interplay between probability and nondeterminism and how modularity simplifies the analysis. We refer to [79] the reader interested in all the details.

6.1 The Consensus Problem

The consensus problem consists of making n asynchronous processes decide on the same value (either 0 or 1) in the presence of stopping faults, given that each process starts with its own initial value. The initial value is provided by the environment during initialization. We say that an algorithm solves the consensus problem if it satisfies the following properties.

Validity: If a process decides on a value within an execution of the algorithm, then this value is the initial value of some process.
Agreement: Any two processes that decide within an execution of the algorithm decide on the same value.
Wait-free termination: All initialized and non-failed processes eventually decide.

It is known from [38] that there is no deterministic algorithm for asynchronous processes that solves consensus and guarantees termination even in the presence of at most one single faulty process. However, the problem becomes solvable using randomization if we relax the termination condition and we replace it with the following condition.

Probabilistic wait-free termination: With probability 1, all initialized and non-failed processes eventually decide.

6.2 Description of the Algorithm

The algorithm of Aspnes and Herlihy proceeds in rounds. Every process maintains a variable with two fields, *value* and *round*, that contain the process' current preferred value ($0, 1$ or $\perp$) and current round (a non-negative integer), respectively. We say that a process is at round r if its *round* field is equal to r. Note that, due to asynchrony, different processes could be at different rounds at some point of an execution. The variables $(value, round)$ are multiple-reader single-writer. Each process starts with its *round* field initialized to 0 and its *value* field initialized to $\perp$.

After receiving the initial value to agree on, each process i executes the following loop. It first reads the $(value, round)$ variables of all other processes in its local memory. We say that process i is a *leader* if according to its readings its own round is greater than or equal to the rounds of all other processes. We

also say that a process i *observed* that another process j is a leader if according to i's readings the round of j is greater than or equal to the rounds of all other processes. If process i at round r discovers that it is a leader, and that according to its readings all processes that are at rounds r and $r-1$ have the same value as i, then i breaks out of the loop and decides on its value. Otherwise, if all processes that i observed to be leaders have the same value v , then i sets its value to v, increments its round and proceeds to the next iteration of the loop. In the remaining case (leaders that i observed do not agree), i sets its value to $\perp$ and scans the other processes again. If once again the leaders observed by i do not agree, then i determines its new preferred value for the next round by invoking a coin flipping protocol. There is a separate coin flipping protocol for each round. Figure 1 gives a high level view of the algorithm. The left box is the main algorithm which is subdivided into processes; the right boxes are the coin flipping protocols which interact with the main algorithm through some invocation and response messages.

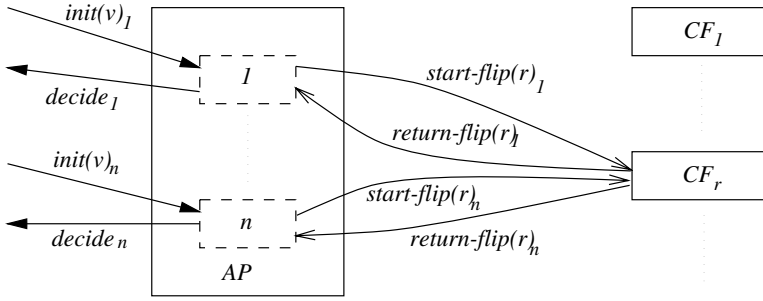


Fig. 1. The interaction diagram of the algorithm of Aspnes and Herlihy.

We represent the main part of the algorithm as an automaton AP (Agreement Protocol), and the coin flipping protocols as probabilistic automata CF_r (Coin Flipper), one for each round r. With this decomposition we can prove several important properties of the algorithm as properties of AP using ordinary techniques for non-probabilistic systems.

6.3 Informal Analysis of the Algorithm

It is easy to show that the algorithm satisfies validity since if all processes start all with the same value v, then no process will ever observe disagreement among the leaders and no process will ever propose a value different from v.

It is more difficult to show that the algorithm satisfies agreement. The first important observation is that agreement does not rely on probability, but rather on the fact that the processes at the two highest rounds all agree when a process decides. The very strict condition on the decision action ensures that no process

will ever be able to compromise a decision that was taken already. If a process decides v at round r, then all processes at round r agree on v and no process at round $r-1$ can observe leaders with values different from v. More precisely, suppose for the sake of contradiction that the decision is taken by process P and that there is a process Q at round $r-1$ that is up to proposing a value different from v for round r or up to flipping a coin for the value to propose at round r. Let Q be the first such process. This means that all processes at round r or higher agree on v. We distinguish two exhaustive cases.

1. Process Q observed that the leaders agree on a value different from v.
   In this case, since all processes at round r prefer v, process Q observed that the leaders are at round $r-1$. Thus, since Q is at round $r-1$, Q itself prefers a value different from v at round $r-1$. Consider the last observation that P made of Q. If P observed Q at round $r-1$, then the value preferred by Q at round $r-1$ must be v, a contradiction (it is possible to show that a process cannot switch its preferred value within a round); if P did not observe Q at round $r-1$, then P was already at round r when Q moved to round $r-1$, which means that Q observed at least one process at round r during its last scan, again a contradiction.
2. Process Q observed that the leaders do not agree on v.
   Since during the second scan of process Q the value proposed by Q is $\perp$, process P observed Q either at a round lower than $r-1$ or while process Q was scanning the other processes for the first time. In both cases during the second scan of round $r-1$ process Q sees that process P is at round r, and thus that all leaders agree on v. This is a contradiction.

The agreement property is quite intricate to analyze, and the analysis above may look incomplete since each statement relies on the understanding of several subtle interactions between processes. However, assuming that all the statements are correct, the informal analysis above provides the main ideas behind the correctness of the algorithm of Aspnes and Herlihy. In the formal proof [79] all the informal analysis above is embedded into an invariant property that can be proved easily by induction. No probability involved in the proof.

The termination property (eventually some process will decide) relies strongly on the properties of the coin flipping protocol. If at a certain round the coin flipping protocol behaves like a *global coin flip*, i.e., like the flip of a unique coin the result of which is returned to each process, then termination occurs within a few rounds. Informally, all the processes that do not flip coins to select the value for the next round will select the same value, and all the processes that flip obtain the same value. The key problem is how to define a coin flipper that behaves like a global coin flipper with high probability. We postpone the discussion to Section 6.5.

6.4 Using Modularity for the Analysis of Termination

We now show how modularity plays a crucial role in the analysis of termination and concentrate on showing that in the algorithm of Aspnes and Herlihy

some decision is reached within some expected number of rounds. This property depends on the probabilistic properties of the coin flipping protocols. However, there are several progress properties of the algorithm that do not depend on any probabilistic assumption. We first study such properties, which are properties of AP only, and then we use modularity to combine these properties with the probabilistic behavior of the coin flippers.

For each round r, let CF_r be a coin flipping protocol, that is, a probabilistic automaton with the interface of a coin flipper of Figure 1. Define AH (Aspnes-Herlihy) to be $AP \parallel CF$, where CF is $(\|_{r \geq 1} CF_r)$.

Define a function $\phi_{MaxRound}$ that takes a finite execution α and gives

$$\phi_{MaxRound}(\alpha) \stackrel{\triangle}{=} lstate(\alpha).max\text{-}round - fstate(\alpha).max\text{-}round,$$

where the *max-round* component of a state is the maximum round number among all the processes. Define the following sets of states.

- $\mathcal{R}$ the set of reachable states of AH such that some process is involved in the consensus protocol;
- $\mathcal{D}$ the set of reachable states of AH such that no process is involved in the consensus protocol (each process either has decided or has not started yet).

We call the states of $\mathcal{R}$ *active*, since they represent situations where some process is participating actively in the consensus protocol. We want to show that, under some special conditions on the coin flipping protocols, starting from any state of $\mathcal{R}$, a state from $\mathcal{D}$ is reached within some bounded number of rounds. It turns out that it is easier to split the problem in two parts: first we show a simple property that, unless the algorithm terminates, the system reaches a point where one process has just moved to a new maximum round ($\mathcal{F}_0$ and $\mathcal{F}_1$ below, where the subscript corresponds to the value preferred by the process at the maximum round); then, we show that from such an intermediate point, under some special conditions on the coin flipping protocols, the algorithm terminates. Formally, define the following sets of states.

- $\mathcal{F}_0$ the set of states of $\mathcal{R}$ where there exists a round r and a process l such that l just reached round r preferring value 0, and no other process has reached round r yet;
- $\mathcal{F}_1$ the set of states of $\mathcal{R}$ where there exists a round r and a process l such that l just reached round r preferring value 1, and no other process has reached round r yet.

We show two properties, the first of which is almost trivial:

M1 If AH is in a state s of $\mathcal{R}$ and all invocations to the coin flippers on non-failing ports get a response, then a state from $\mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}$ is reached within one round.

M2 If AH is in a state s of $\mathcal{F}_v$, all invocations to the coin flippers on non-failing ports get a response, and all invocations to $CF_{s.max\text{-}round}$ get only response v, then a state from $\mathcal{D}$ is reached within two rounds.

Informally, "all invocations to the coin flippers on non-failing ports get responses" means that whenever a process invokes a coin flipper, either it fails or it gets a response, and "all invocations to CF_r get only response v" means that whenever CF_r returns a result, the result is v. We are not interested in the formal statements for the purpose of this paper. The proofs of **M1** and **M2** follow standard arguments for nondeterministic systems based on invariants.

In Section 6.5 we show how to build a distributed implementation of the coin flippers that satisfies the following properties.

C1 For each probabilistic execution of CF_r that starts with a reachable state of CF_r, the probability of the executions where all invocations to the coin flippers on non-failing ports get responses is 1

C2 For each fair probabilistic execution of CF_r, and each value $v \in \{0, 1\}$, the probability of the executions where all invocations to CF_r get only response v is at least p, where p is a real number in $[0, 1]$.

We use modularity to combine **M1**, **M2**, **C1** and **C2** and show that AH guarantees progress within expected $O(1/p)$ rounds. That is, we prove the following proposition.

Proposition 1. *If each coin flipping protocol CF_r satisfies properties* **C1** *and* **C2***, then in AH, starting from any state of $\mathcal{R}$ and under any scheduler, a state from $\mathcal{D}$ is reached within at most $O(1/p)$ expected rounds.*

The main statement that we use in the proof is

$$\mathcal{R} \xrightarrow[p]{\phi_{MaxRound} \leq 3} \mathcal{D}. \tag{1}$$

whose meaning is that under any scheduler, starting from a state of $\mathcal{R}$, with probability at least p a state from $\mathcal{D}$ is reached within 3 rounds. Then we simply have to iterate the experiment of (1) until the experiment is successful, visiting at most $3/p$ expected rounds. To prove Statement (1) we prove two intermediate statements:

$$\mathcal{R} \xrightarrow[1]{\phi_{MaxRound} \leq 1} \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}, \tag{2}$$

and for each $v \in \{0, 1\}$,

$$\mathcal{F}_v \xrightarrow[p]{\phi_{MaxRound} \leq 2} \mathcal{D}. \tag{3}$$

The proofs of Statements (2) and (3) rely on the properties **M1**, **M2**, **C1**, and **C2** and on Theorem 1.

Proposition 2. *Assuming that the coin flippers in AH satisfy* **C1***,*

$$\mathcal{R} \xrightarrow[1]{\phi_{MaxRound} \leq 1} \mathcal{F}_1 \cup \mathcal{F}_0 \cup \mathcal{D}. \tag{4}$$

Proof. Let μ be a probabilistic execution of AH that starts from a state of $\mathcal{R}$. Consider the inverse image θ of the event in $\pi_{CF}(\mu)$ stating that each invocation on a non failing port gets an answer. Then, the probability of θ is 1 by **C1** and Theorem 1. Observe that each execution of $\pi_{AP}(\theta)$ satisfies the premise of **M1**. Thus, in each execution of θ a state from $\mathcal{F}_1 \cup \mathcal{F}_0 \cup \mathcal{D}$ is reached within 1 round.

Proposition 3. *Assuming that the coin flippers in AH satisfy* **C1** *and* **C2**,

$$\mathcal{F}_v \xrightarrow[p]{\phi_{MaxRound} \leq 2} \mathcal{D}. \tag{5}$$

Proof. Let μ be a probabilistic execution of AH that starts from a state s_0 of $\mathcal{F}_v$, and let $r = s_0.max\text{-}round$. Consider the inverse image θ of the event in $\pi_{CF}(\mu)$ stating that each invocation on a non failing port gets an answer and each response of CF_r is v. Then, the probability of θ is at least p by **C1**, **C2** and Theorem 1. Observe that $\pi_{AP}(\theta)$ satisfies the premises of **M1** and **M2**. Thus, in each execution of θ a state from $\mathcal{D}$ is reached within 2 rounds.

6.5 The Coin Flippers

We are left to show how to build a distributed coin flipping protocol with the properties **C1** and **C2**. In this section we build an almost distributed version of the coin flipping protocol where processes interact through a multiple-writer multiple-reader shared register; in Section 6.6 we refine the protocol of this section to yield a distributed protocol. The protocol is based on random walks and satisfies properties **C1** and **C2** with a sufficiently high probability p that is independent of n.

We represent the coin flipping protocol by letting an automaton DCN_r (Distributed CoiN) interact with a centralized counter CT_r (CounTer), that is, $CF_r = Hide_I(DCN_r \parallel CT_r)$, where I is the set of actions used for the interaction between DCN_r and CT_r, and $Hide_I$ is an operator that transforms the actions of I from external to internal. Figure 2 shows the structure of the coin flipping protocol. In this section, DCN_r is distributed while CT_r is composed of n processes that receive requests from DCN_r and read/update a single shared variable: the details of the distributed implementation of a shared counter are not necessary for any properties of the coin flipping protocol. The algorithm is simple. To flip a coin a process reads the value of the counter and returns 0 if the counter is below $-Kn$, K being a constant, 1 if the counter is above Kn. If the counter is between $-Kn$ and Kn, then the process flips a coin to decide whether to increment or decrement the shared counter and then starts from the beginning.

Since the protocols for DCN_r and CT_r are the same for any round r, we drop the subscript r from our notation.

The idea behind the coin flipping protocol is very simple: the difference between the heads and tails obtained in the elementary coin flips form a stochastic process which is known in the literature under the name of random walk [37]. The value of the shared counter and the actual difference between heads and tails may differ by at most n since in the worst case each process may be trying to update the counter. Finally, if the difference between heads and tails is greater than or equal to $(K + 1)n$, then no process will ever observe a value below Kn. This last property requires a careful analysis, but the idea is that the processes that have to update the counter will not flip any more and the other processes will flip at most once.

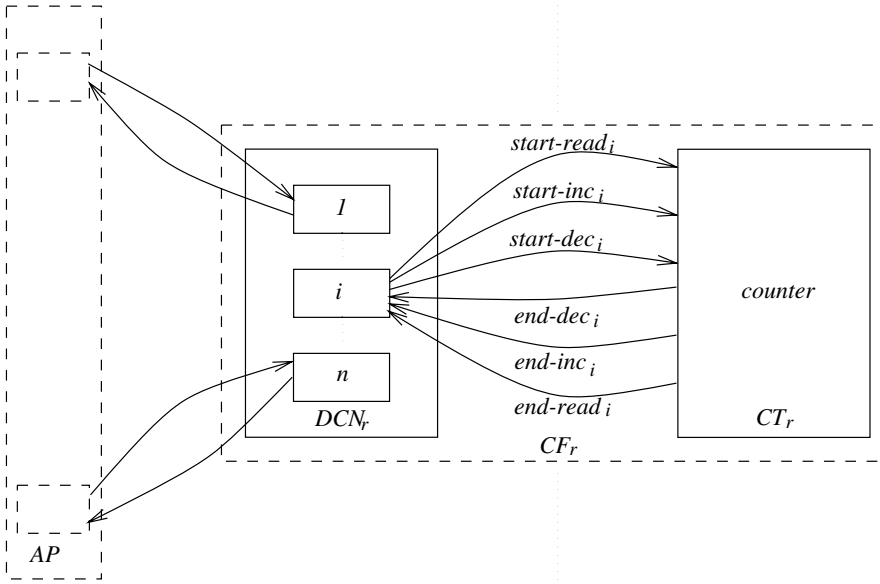


Fig. 2. The structure of the coin flipping protocol.

Based on the properties above, if the difference between heads and tails ends above $(K+1)n$ before ending below $-(K-1)n$, then all processes will return head: no process will ever observe a value below $-Kn$ since the value of the counter and the difference between heads and tails differ by at most n; furthermore, after hitting $(K+1)n$ all processes will observe a value above Kn. A symmetric argument holds for tail. From random walk theory, the barrier $(K+1)n$ is reached before $-(K-1)n$ with probability $(K-1)/2K$.

Most of the properties described in this informal analysis do not rely on any probabilistic assumption, and thus will not be described in detail in this paper. The only thing that is relevant here is the kind of coin lemma that we need to use for the random walk. Essentially, in the spirit of coin lemmas, we consider the set θ of those executions where either there are finitely many coin flips and the barrier $-(K-1)n$ is not reached (i.e., it is possible to fix the values of the unflipped coins so that the barrier $(K+1)n$ is reached before $-(K-1)n$), or the barrier $(K+1)n$ is reached before $-(K-1)n$. Then we are guaranteed that in each probabilistic execution the set Θ has probability at least $(K-1)/2K$. In the analysis of the coin flippers the case where the barriers are not reached does occur whenever all the processes that are flipping fail. However, such event does not compromise termination.

6.6 Implementing the Shared Counter

In this section we describe how to obtain an implementation of CT that can replace the abstract automaton CT in CF without compromising properties **C1**

and **C2** with $p = (K-1)/2K$. In this way, using the coin flipping protocol with the new counter, we obtain a protocol for consensus that uses only single-writer multiple-reader shared variables.

The implementation of CT, which we denote by DCT (Distributed CounTer), is an adaptation of an algorithm proposed by Lamport [61] for read/write registers. For our purpose the details of DCT are completely irrelevant. The important fact is that using standard results about atomic objects [66], we can find a simulation relation from DCT to CT. Since simulation relations are preserved by composition (cf. Theorem 2), we have a simulation relation between the whole algorithm of Aspnes and Herlihy with the distributed counter and the whole algorithm with the centralized counter, which implies trace distribution inclusion as well.

Since the termination property is a property expressible by means of trace distributions (simply perform some visible action whenever a decision is taken), trace distribution inclusion is sufficient to show that the algorithm of Aspnes and Herlihy with the distributed counter works correctly within an expected constant number of rounds.

7 Performance Evaluation with Complexity Functions

Although it is not possible to estimate the running time of a distributed asynchronous algorithm (processor speeds are unknown), it is possible to give an estimate of the running time under some assumptions on the minimum speed of the processors. Typically we assume that each processor completes a computational step within 1 time unit once the computational step can be performed; any other time limit can be studied simply by scaling.

In the randomized case things are complicated by the fact that our performance measures are expectations and that such expectations change depending on how the nondeterminism is resolved. Furthermore, it is easier to compute expected rounds rather than expected time. Once again, the main idea is to work in the absence of probabilities and then lift the results to expectations. We start with the notion of a complexity function.

7.1 Complexity Functions

Fix a probabilistic automaton $\mathcal{A}$. A *complexity function* is a function from executions of $\mathcal{A}$ to $\Re^{\geq 0}$. A *complexity measure* is a complexity function ϕ such that, for each pair α_1 and α_2 of executions that can be concatenated, $max(\phi(\alpha_1), \phi(\alpha_2)) \leq \phi(\alpha_1 \frown \alpha_2) \leq \phi(\alpha_1) + \phi(\alpha_2)$.

Informally, a complexity measure is a function that determines the complexity of an execution. A complexity measure satisfies two natural requirements: the complexity of two tasks performed sequentially should not exceed the complexity of performing the two tasks separately and should be at least as large as the complexity of the more complex task: it should not be possible to accomplish more by working less. The restrictions imposed on complexity measures are required for the iterative argument of Section 6.4.

7.2 Complexity Functions and Random Variables

Consider a probabilistic execution μ of $\mathcal{A}$ and a set of executions Θ such that $\mu[\cup_{q\in\Theta} C_q] = 1$. The set Θ is said to be a full cut of μ and the event denoted by Θ is said to be *finitely satisfiable*. Informally, the elements of Θ represent the points where the property denoted by Θ is satisfied. Let ϕ be a complexity function. Then the restriction of ϕ to Θ is a random variable from the measurable space $(\Theta, 2^{\Theta})$. Define the measure on Θ as $P[q] = \mu[C_q]$. We call such random variable the random variable induced by Θ. We can define the expected complexity ϕ to reach Θ in μ as follows:

$$E_{\mu,\Theta}[\phi] \triangleq \sum_{q\in\Theta} \phi(q)\mu[C_q],$$

i.e., the expected complexity ϕ to reach Θ in μ is the expected value of the random variable induced by Θ.

7.3 Linear Combination of Complexity Functions

Given the close relationship between complexity functions and random variables in the presence of full cuts, typical results for random variables can be used. In particular, if several complexity measures are related by a linear inequality, then their expected values over a full cut are related by the same linear inequality. We use this property for the time analysis of the protocol of Aspnes and Herlihy. That is, we express the time complexity of the protocol in terms of two other complexity measures (rounds and elementary coin flips), and then we use Proposition 4 below to derive an upper bound on the expected time for termination based on upper bounds on the expected values of the other two complexity measures. The analysis of the other two complexity measures is simpler, and the relationship between time and the other two complexity measures can be studied using known methods for ordinary nondeterministic systems, with no probability involved.

Proposition 4. *Let μ be a probabilistic execution of some probabilistic automaton $\mathcal{A}$, and let Θ be a full cut of μ. Let ϕ, ϕ_1, ϕ_2 be complexity functions, and c_1, c_2 be two constants such that, for each $\alpha \in \Theta$, $\phi(\alpha) \leq c_1\phi_1(\alpha) + c_2\phi_2(\alpha)$. Then $E_{\mu,\Theta}[\phi] \leq c_1 E_{\mu,\Theta}[\phi_1] + c_2 E_{\mu,\Theta}[\phi_2]$.* ■

7.4 Modular Analysis with Complexity Functions

To verify properties in a modular way it is useful to derive complexity properties of a composite systems based on complexity properties of the single components. Proposition 5 helps in doing this. Informally, suppose that we have a complexity function ϕ for $\mathcal{A} = \mathcal{A}_1 \,\|\, \mathcal{A}_2$ and a complexity function ϕ_1 for $\mathcal{A}_1$ such that ϕ and ϕ_1 coincide up to projection. In other words ϕ measures in $\mathcal{A}$ the property of $\mathcal{A}_1$ that is measured by ϕ_1. Furthermore, suppose that we know an upper bound

on the expected value of ϕ_1 that is independent of how the nondeterminism is resolved in $\mathcal{A}_1$. Then, the same upper bound is valid for ϕ as well. In other words, the property that we know about $\mathcal{A}_1$ can be lifted to $\mathcal{A}$.

Proposition 5. *Let $\mathcal{A}$ be $\mathcal{A}_1 \parallel \mathcal{A}_2$, and let $i \in \{1,2\}$. Let ϕ be a complexity function for $\mathcal{A}$, and let ϕ_i be a complexity function for $\mathcal{A}_i$. Suppose that for each finite execution α of $\mathcal{A}$, $\phi(\alpha) = \phi_i(\alpha\lceil\mathcal{A}_i)$. Let c be a constant. Suppose that for each probabilistic execution μ of $\mathcal{A}_i$ and each full cut Θ of μ, $E_{\mu,\Theta}[\phi_i] \leq c$. Then, for each probabilistic execution μ of $\mathcal{A}$ and each full cut Θ of μ, $E_{\mu,\Theta}[\phi] \leq c$.* ■

8 Performance of the Randomized Consensus Algorithm

For the analysis of the algorithm of Aspnes and Herlihy we define several complexity functions and prove linear inequalities about them. Then we lift the results to expectations using the property of reaching $\mathcal{D}$ as our full cut. The complexity measures are the following:

- ϕ_t: Time elapsed.
- $\phi_{\mathit{flip},r}$: Elementary coin flips at round r.
- $\phi_{id,r}$: Updates to the counter at round r.
- ϕ_{id}: Updates to the counters.

Denote by *DAH* the algorithm of Aspnes and Herlihy with the distributed shared counter, and denote by DCF_r the coin flipper for round r with the distributed shared counter. A property that can be proved by means of ordinary nondeterministic analysis is the following.

Lemma 2. *Let α be an execution of DAH where each computational step is taken within 1 time unit, and let $R = \mathit{fstate}(\alpha).\mathit{max\text{-}round}$. Suppose that all the states of α, with the possible exception of $\mathit{lstate}(\alpha)$ are active, that is, are states of $\mathcal{R}$. Then, $\phi_t(\alpha) \leq d_1 n^2(\phi_{MaxRound}(\alpha)+R)+d_2 n\phi_{id}(\alpha)+d_3 n^2$ for some constants d_1, d_2, and d_3.*

That is, during the consensus protocol we perform quadratic work for each round and linear work for each update to the shared counters. We know already from Section 6.4 that the expected number of new rounds is constant. We need to derive an upper bound on ϕ_{id}.

Since within a coin flipper the number of updates to the shared counter differs from the number of flipped coins at most by n, we can state that for each $r > 0$, $\phi_{\mathit{flip},r} \leq \phi_{id,r} + n$. Using a coin lemma like argument and the relative properties of random walks, we can show that the expected value of $\phi_{\mathit{flip},r}$ is quadratic in n. Thus, by Proposition 5, for each probabilistic execution μ and each full cut Θ, $E_{\mu,\Theta}[\phi_{id,r}] = O(n^2)$.

Since the expected number of visited rounds is R plus a constant number, we obtain that $E_{\mu,\Theta}[\phi_{id}] = O(Rn^2)$. By replacing in the expression of Lemma 2, we obtain that the expected time for termination is $O(Rn^3)$. In particular, if initially $R = 0$, the expected time for termination is $O(n^3)$.

Once again we refer to [79] the reader interested in the details of the proof.

9 Concluding Remarks

In this paper we have illustrated some of the techniques that can be used for the analysis of randomized distributed algorithms. We addressed the main problem of handling probability in the presence of nondeterminism, an intrinsic characteristic of concurrent systems. All the techniques that we have illustrated originate from concurrency theory and from the experience that we have gained through the analysis of concurrent systems that do not involve probability.

We believe that this area constitutes one of the possible bridges between concurrency theory and the area of performance analysis. The results of this paper should serve two purposes: migrating some of the typical techniques of concurrency theory to performance evaluation, a goal that is achieved also by stochastic process algebras, and exposing people familiar with performance evaluation to the typical problems of concurrent systems, so that the converse migration can take place.

It would be interesting to identify other areas of application that go beyond randomized distributed algorithms to see whether the techniques of this paper, possibly extended and generalized, can be applied. It would be useful as well to see how techniques based on rewards would extend and or adapt to coin lemma based arguments.

Finally, computer aided verification is getting increasing attention. We are currently involved in a project where we use a model checker and a theorem prover to get mechanized proofs for randomized distributed algorithms [58]. In this area it is very important to find the right balance between the amount of human intervention (the problem is undecidable in general and in any case it has a high complexity) and the parts of an analysis that can be carried out via model checking and/or theorem proving. Some information about related topics can be found in this volume [55].

References

1. Y. Afek and Y. Matias. Simple and efficient election algorithms for anonymous networks. In *3rd International Workshop on Distributed Algorithms,* Nice, France, 1989.
2. S. Aggarwal. Time optimal self-stabilizing spanning tree algorithms. Technical Report MIT/LCS/TR-632, MIT Laboratory for Computer Science, 1994. Master's thesis.
3. S. Aggarwal and S. Kutten. Time optimal self stabilizing spanning tree algorithms. In R.K. Shyamasundar, editor, *13th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, pages 400–410, Bombay, India., December 1993. Springer-Verlag.
4. L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997. Available as Technical report STAN-CS-TR-98-1601.
5. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proceedings* 13^{th} *Annual Symposium on Logic in Computer Science*, pages 454–465. IEEE Computer Society Press, 1998.

6. L. de Alfaro. Stochastic transition systems. In *Proceedings of CONCUR 98,* Nice, France, volume 1466 of *Lecture Notes in Computer Science*, pages 423–438. Springer-Verlag, 1998.
7. L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In L. Baeten and S. Mauw, editors, *Proceedings of CONCUR 99,* Eindhoven, The Netherlands, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1999.
8. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using mtbdds and the kronecker representation. In *Proceedings of TACAS'2000*, volume 1785 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.
9. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for probabilistic real-time systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings* 18^{th} *ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 115–136. Springer-Verlag, 1991.
10. R. Alur, C. Courcoubetis, and D.L. Dill. Verifying automata specifications of probabilistic real-time systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 28–44. Springer-Verlag, 1991.
11. S. Andova. Process algebra with probabilistic choice. In J.P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems*, number 1601 in Lecture Notes in Computer Science, pages 111–129. Springer-Verlag, 1999.
12. J. Aspnes and M.P. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, September 1990.
13. James Aspnes and Orli Waarts. Randomized consensus in expected $O(n \log^2 n)$ operations per processor. In *33rd Annual Symposium on Foundations of Computer Science*, pages 137–146, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.
14. J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 122:234–255, 1995.
15. J.C.M. Baeten and J.W. Klop, editors. *Proceedings of CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
16. J.C.M. Baeten and W.P Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.
17. C. Baier. On algorithmic verification methods for probabilistic systems, 1998. Habilitation thesis, University of Mannheim.
18. C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. Technical Report 99/12, Centre for Telematics and Information Technology, University of Twente, 1999.
19. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11, 1998.
20. C. Baier and M. Stoelinga. Norm functions for bisimulations with delays. In *Proceedings of FOSSACS*. Springer-Verlag, 2000.
21. M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In *Proceedings of the* 2^{nd} *Annual ACM Symposium on Principles of Distributed Computing,* Montreal, Quebec, Canada, August 1983.
22. M. Bernardo, L. Donatiello, and R. Gorrieri. Modeling and analyzing concurrent systems with MPA. In U. Herzog and M. Rettelbach, editors, *Proceedings of the Second Workshop on Process Algebras and Performance Modelling* (PAPM), Erlangen, Germany, pages 175–189, 1994.

23. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings Conferenco on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513, 1995.
24. B. Bloom and A. Meyer. A remark on bisimulation between probabilistic processes. In *Proceedings of the Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*, pages 26–40, 1989.
25. E. Brinksma and H. Hermanns. Process algebra and stochastic process algebra. This volume.
26. I. Christoff. *Testing Equivalences for Probabilistic Processes.* PhD thesis, Department of Computer Science, Uppsala University, 1990.
27. R. Cleaveland, S.A. Smolka, and A. Zwarico. Testing preorders for probabilistic processes (extended abstract). In *Proceedings* 19^{th} *ICALP,* Madrid, volume 623 of *Lecture Notes in Computer Science*, pages 708–719. Springer-Verlag, 1992.
28. W.R. Cleaveland, editor. *Proceedings of CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.
29. C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In 29^{th} *Annual Symposium on Foundations of Computer Science*, pages 338–345, 1988.
30. C. Courcoubetis and M. Yannakakis. Markov decision procedures and regular events. In M. Paterson, editor, *Proceedings* 17^{th} *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer-Verlag, July 1990.
31. P. D'Argenio, H. Hermanns, and J.P. Katoen. On asynchronous generative parallel composition. In *Proceedings of PROBMIV'98*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1999.
32. C. Derman. *Finite State Markovian Decision Processes.* Acedemic Press, 1970.
33. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating continuous Markov processes. In *Proceedings* 15^{th} *Annual Symposium on Logic in Computer Science,* Santa Barbara, California, pages 95–106. IEEE Computer Society Press, 2000.
34. J. Desharnais, V. Gupta, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 1999.
35. S. Dolev, A. Israeli, and S. Moran. Analyzing expected time by scheduler-luck games. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, April 1997.
36. E.A. Emerson and E.C. Clarke. Using branching time temporal logic to synthesize synchronous skeletons. *Science of Computer Programming*, 2:241–266, 1982.
37. W. Feller. *An Introduction to Probability Theory and its Applications. Volume 1.* Jokn Wiley & Sons, Inc., 1950.
38. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with a family of faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
39. A. Giacalone, C.C Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods (IFIP TC2),* Sea of Galilee, Israel, 1990.
40. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1996.
41. R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings* 5^{th} *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 130–141. IEEE Computer Society Press, 1990.

42. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems. Joint Tutorial Papers of Performance '93 and Sigmetrics '93*, volume 729 of *Lecture Notes in Computer Science*, pages 121–146. Springer-Verlag, 1993.
43. H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Science, Uppsala University, 1991.
44. H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems.* Elsevier, 1994.
45. H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proceedings of the* 10^{th} *IEEE Symposium on Real-Time Systems,* Santa Monica, Ca., 1989.
46. H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the* 11^{th} *IEEE Symposium on Real-Time Systems,* Orlando, Fl., 1990.
47. J. Hillston. Exploiting structure in solution: decomposing compositional models. This volume.
48. J. Hillston. PEPA: Performance enhanced process algebra. Technical Report CSR-24-93, Department of Computer Science, University of Edimburgh (UK), 1993.
49. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.
50. C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings* 4^{th} *Annual Symposium on Logic in Computer Science,* Asilomar, California, pages 186–195. IEEE Computer Society Press, 1989.
51. B. Jonsson. Simulations between specifications of distributed systems. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*, pages 346–360. Springer-Verlag, 1991.
52. B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, July 1991.
53. B. Jonsson and J. Parrow, editors. *Proceedings of CONCUR 94,* Uppsala, Sweden, volume 836 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.
54. C.C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In Baeten and Klop [15], pages 367–383.
55. J.-P. Katoen and P.R. D'Argenio. General distributions in process algebra. This volume.
56. R. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):561–572, 1976.
57. E. Kushilevitz and M. Rabin. Randomized mutual exclusion algorithms revisited. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, pages 275–284, 1992.
58. M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In *Proceedings of the* 13^{th} *Workshop on Computer Aided Verification,* Paris, France, July 2001, Lecture Notes in Computer Science. Springer-Verlag, 2001.
59. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic real-time graphs. In Palamidessi [74], pages 132–137.

60. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 286, 2001.
61. L. Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, 1977.
62. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Conference Record of the* 16^{th} *ACM Symposium on Principles of Programming Languages,* Austin, Texas, pages 344–352, 1989.
63. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
64. K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In Cleaveland [28], pages 456–471.
65. D. Lehmann and M. Rabin. On the advantage of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the* 8^{th} *Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.
66. N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., 1996.
67. N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the* 13^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Los Angeles, CA, pages 314–323, 1994.
68. N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, Canada, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.
69. Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.
70. A. McIver. Reasoning about efficiency within a probabilistic mu-calculus. In *Proceedings of PROBMIV'98*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1999.
71. R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.
72. Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.
73. M. Nunez and D. de Frutos. Testing semantics for probabilistic LOTOS. In *Proceedings of Formal Description Techniques VIII*, pages 365–380, 1995.
74. C. Palamidessi, editor. *Proceedings of CONCUR 2000,* University Park, PA, USA, volume 1877 of *Lecture Notes in Computer Science.* Springer-Verlag, 2000.
75. A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In Palamidessi [74], pages 334–349.
76. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
77. A. Pnueli and L. Zuck. Probabilistic verification. *Information and Computation*, 1(1):1–29, 1993.
78. A. Pogosyants and R. Segala. Formal verification of timed properties of randomized distributed algorithms. In *Proceedings of the* 14^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Ottawa, Ontario, Canada, pages 174–183, August 1995.

79. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. *Distributed Computing*, 13:155–186, July 2000.
80. M.O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.
81. M.O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Results*, pages 21–39. Academic Press, 1976.
82. M.O. Rabin. N-process mutual exclusion with bounded waiting by $4 \log N$ shared variables. *Journal of Computer and System Sciences*, 25:66–75, 1982.
83. J.R. Rao. Reasoning about probabilistic algorithms. In *Proceedings of the* 9^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1990.
84. I. Saias. Proving probabilistic correctness: the case of Rabin's algorithm for mutual exclusion. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1992.
85. R. Segala. A compositional trace-based semantics for probabilistic automata. In I. Lee and S.A. Smolka, editors, *Proceedings of CONCUR 95,* Philadelphia, PA, USA, volume 962 of *Lecture Notes in Computer Science*, pages 234–248. Springer-Verlag, 1995.
86. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995. Also appears as technical report MIT/LCS/TR-676.
87. R. Segala. Testing probabilistic automata. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR 95,* Pisa, Italy, volume 1119 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 1996.
88. R. Segala. The essence of coin lemmas. In *Proceedings of PROBMIV'98*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1999.
89. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In Jonsson and Parrow [53], pages 481–496.
90. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
91. K. Seidel. Probabilistic communicating processes. Technical Report PRG-102, Ph.D. Thesis, Programming Research Group, Oxford University Computing Laboratory, 1992.
92. C. Tofts. A synchronous calculus of relative frequencies. In Baeten and Klop [15].
93. M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, Portland, OR, 1985.
94. S.H. Wu, S. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. In Jonsson and Parrow [53].
95. S.H. Wu, S. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1-2):1–38, 1999.
96. W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Protocol Specification, Testing and Verification XII*, pages 47–61, 1992.

Constructing Automata from Temporal Logic Formulas : A Tutorial*

Pierre Wolper

Université de Liège,
Institut Montefiore, B28,
4000 Liège, Belgium
pw@montefiore.ulg.ac.be,
http://www.montefiore.ulg.ac.be/~pw/

Abstract. This paper presents a tutorial introduction to the construction of finite-automata on infinite words from linear-time temporal logic formulas. After defining the source and target formalisms, it describes a first construction whose correctness is quite direct to establish, but whose behavior is always equal to the worst-case upper bound. It then turns to the techniques that can be used to improve this algorithm in order to obtain the quite effective algorithms that are now in use.

1 Introduction

Model checking [3,11,16] is a widespread technique for verifying temporal properties of reactive programs. There are several ways to develop the theory of model checking, a particularly attractive one being through the construction of automata from temporal logic formulas [16,2]. As a result, there has been a fair amount of interest in the construction of automata from temporal logical formulas, the history of which is actually fairly interesting.

The starting point is clearly the work of Büchi on the decidability of the first and second-order monadic theories of one successor [1]. These decidability results were obtained through a translation to infinite-word automata, for which Büchi had to prove a very nontrivial complementation lemma. The translation is nonelementary, but this is the best that can be done. It is quite obvious that linear-time temporal logic can be translated to the first-order theory of one successor and hence to infinite-word automata. From a logician's point of view, this could be seen as settling the question, but an interest in using temporal logic for computer science applications, in particular program synthesis [10,5] triggered a second look at the problem. Indeed, it was rather obvious that a nonelementary construction was not necessary to build an automaton from a temporal logic formula; it could be done within a single exponential by a direct

* This work was partially funded by a grant of the "Communauté française de Belgique - Direction de la recherche scientifique - Actions de recherche concertées".

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 261–277, 2001.

construction [18,17]. As originally presented, this construction was worst and best case exponential. Though it was fairly clear that it could be modified to operate more effectively on many instances, nothing was written about this, probably because the topic was thought to be rather trivial and had no bearing on general complexity results.

Nevertheless, the idea of doing model checking through the construction of automata was taken seriously, at least by some, and attempts were made to incorporate automata-theoretic model checking into tools, notably into SPIN [7,8]. Of course, this required an effective implementation of the logic to automaton translation algorithm and the pragmatics of doing this are not entirely obvious. A description of such an implementation was given in [6] and improved algorithms have been proposed since [4,13]. Note that there are some questions about how to measure such improvements since the worst-case complexity of the algorithms stays the same. Nevertheless, experiments show that, for the temporal logic formulas most frequently used in verification, the automata can be kept quite small. Thus, even though it is an intrinsically exponential process, building an automaton from a temporal logic formula appears to be perfectly feasible in practice. What is surprising is that it took quite a long time for the details of a usable algorithmic solution to be developed and codified.

The goal of this paper is to provide a tutorial introduction to the construction of Büchi infinite-word automata from linear temporal logic formulas. After an introduction to temporal logic and a presentation of infinite-word automata that stresses their kinship to logic, a first simple, but always exponential, construction is presented. This construction is similar to the one of [18,17], but is more streamlined since it does not deal with the extended temporal logic considered in the earlier work. Thereafter, it is shown how this construction can be adapted to obtain a more effective construction that only builds the needed states of the automaton, as described in [6] and further improved in [4,13].

2 Linear-Time Temporal Logic

Linear-time temporal logic is an extension of propositional logic geared to reasoning about infinite sequences of states. The sequences considered are isomorphic to the natural numbers and each state is a propositional interpretation. The formulas of the logic are built from atomic propositions using Boolean connectives and temporal operators. Purely propositional formulas are interpreted in a single state and the temporal operators indicate in which states of a sequence their arguments must be evaluated.

Formally, the formulas of linear-time temporal logic (LTL) built from a set of atomic propositions P are the following:

- **true**, **false**, p, and $\neg p$, for all $p \in P$;
- $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are LTL formulas;
- $\bigcirc \varphi_1$, $\varphi_1 \, U \, \varphi_2$, and $\varphi_1 \, \tilde{U} \, \varphi_2$, where φ_1 and φ_2 are LTL formulas.

The operator $\bigcirc$ is read "next" and means in the *next* state. The operator U is read "until" and requires that its first argument be true *until* its second

argument is true, which is required to happen. The operator $\tilde{U}$ is the dual of U and is best read as "releases", since it requires that its second argument always be true, a requirement that is *released* as soon as its first argument becomes true. Two derived operators are in very common use. They are

- $\Diamond\,\varphi = \mathbf{true}\,U\,\varphi$, which is read "eventually" and requires that its argument be true *eventually*, i.e. at some point in the future; and
- $\Box\,\varphi = \mathbf{false}\,\tilde{U}\,\varphi$, which is read "always" and requires that its argument be true *always*, i.e. at all future points.

Formally, the semantics of LTL is defined with respect to sequences $\sigma : \mathbf{N} \to 2^P$. For a sequence σ, σ^i represents the suffix of σ obtained by removing its i first states, i.e. $\sigma^i(j) = \sigma(i+j)$. The truth value of a formula on a sequence σ, which is taken to be the truth value obtained by starting the interpretation of the formula in the first state of the sequence, is given by the following rules:

- For all σ, we have $\sigma \models \mathbf{true}$ and $\sigma \not\models \mathbf{false}$;
- $\sigma \models p$ for $p \in P$ iff $p \in \sigma(0)$;
- $\sigma \models \neg p$ for $p \in P$ iff $p \notin \sigma(0)$;
- $\sigma \models \varphi_1 \wedge \varphi_2$ iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$;
- $\sigma \models \varphi_1 \vee \varphi_2$ iff $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$;
- $\sigma \models \bigcirc\,\varphi_1$ iff $\sigma^1 \models \varphi_1$;
- $\sigma \models \varphi_1\,U\,\varphi_2$ iff there exists $i \geq 0$ such that $\sigma^i \models \varphi_2$ and for all $0 \leq j < i$, we have $\sigma^j \models \varphi_1$;
- $\sigma \models \varphi_1\,\tilde{U}\,\varphi_2$ iff for all $i \geq 0$ such that $\sigma^i \not\models \varphi_2$, there exists $0 \leq j < i$ such that $\sigma^j \models \varphi_1$.

In the logic we have defined, negation is only applied to atomic propositions. This restriction can be lifted with the help of the following relations, which are direct consequences of the semantics we have just given:

$$\sigma \not\models \varphi_1\,U\,\varphi_2 \text{ iff } \sigma \models (\neg\varphi_1)\,\tilde{U}(\neg\varphi_2)$$

$$\sigma \not\models \varphi_1\,\tilde{U}\,\varphi_2 \text{ iff } \sigma \models (\neg\varphi_1)\,U(\neg\varphi_2)$$

$$\sigma \not\models \bigcirc\,\varphi_1 \text{ iff } \sigma \models \bigcirc\,\neg\varphi_1.$$

To easily understand the link between temporal logic formulas and automata, it is useful to think of a temporal formula as being a description of a set of infinite sequences: those that satisfy it. Note that to check that a sequence satisfies a temporal logic formula φ, a rather natural way to proceed is to attempt to label each state of the sequence with the subformulas of φ that are true there. One would proceed outwards, starting with the propositional subformulas, and adding exactly those subformulas that are compatible with the semantic rules. Of course, for an infinite sequence, this cannot be done effectively. However, this abstract procedure will turn out to be conceptually very useful.

3 Automata on Infinite Words

Infinite words (or ω-words) are sequences of symbols isomorphic to the natural numbers. Precisely, an infinite word over an alphabet Σ is a mapping $w : \mathbf{N} \to \Sigma$.

An automaton on infinite words is a structure that defines a set of infinite words. Even though infinite word automata look just like traditional automata, one gets a better understanding of them by not considering them as operational objects but, rather, by seeing them as descriptions of sets of infinite sequences, and hence as a particular type of logical formula.

We will consider Büchi and generalized Büchi automata on infinite words. A Büchi infinite word automaton has exactly the same structure as a traditional finite word automaton. It is a tuple $A = \{\Sigma, S, \delta, S_0, F\}$ where

- Σ is an alphabet,
- S is a set of states,
- $\delta : S \times \Sigma \to S$ (deterministic) or $\delta : S \times \Sigma \to 2^S$ (nondeterministic) is a transition function,
- $S_0 \subseteq S$ is a set of initial states (a singleton for deterministic automata), and
- $F \subseteq S$ is a set of accepting states.

What distinguishes a Büchi infinite-word automaton from a finite word automaton is that its semantics are defined over infinite words. Let us now examine these semantics using a somewhat logical point of view. A word w is accepted by an automaton $A = \{\Sigma, S, \delta, S_0, F\}$ (the word satisfies the automaton) if there is a labeling

$$\rho : \mathbf{N} \to S$$

of the word by states such that

- $\rho(0) \in S_0$ (the initial label is an initial state),
- $\forall 0 \leq i, \rho(i+1) \in \delta(\rho(i), w(i))$ (the labeling is compatible with the transition relation),
- $\inf(\rho) \cap F \neq \emptyset$ where $\inf(\rho)$ is the set of states that appear infinitely often in ρ (the set of repeating states intersects F).

Example 1. The automaton of Figure 1 accepts all words over the alphabet $\Sigma = \{a, b\}$ that contain b infinitely often.

Generalized Büchi automata differ from Büchi automata by their acceptance condition. The acceptance condition of a generalized Büchi automaton is a set of sets of states $\mathcal{F} \subseteq 2^S$, and the requirement is that some state of each of the sets $F_i \in \mathcal{F}$ appears infinitely often. More formally, a generalized Büchi $A = \{\Sigma, S, \delta, S_0, \mathcal{F}\}$ accepts a word w if there is a labeling ρ of w by states of A that satisfies the same first two conditions as given for Büchi automata, the third being replaced by:

- For each $F_i \in \mathcal{F}$, $\inf(\rho) \cap F_i \neq \emptyset$.

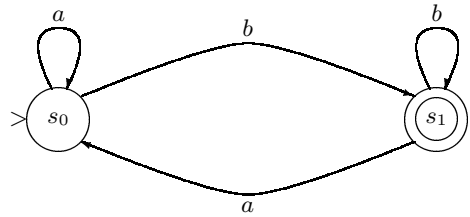


Fig. 1. An automaton accepting all words over $\Sigma = \{a, b\}$ containing b infinitely often

As the following lemma shows, generalized Büchi automata accept exactly the same languages as Büchi automata.

Lemma 1. *Given a generalized Büchi automaton, one can construct an equivalent Büchi automaton.*

Proof. Given a generalized Büchi automaton $A = (\Sigma, S, \delta, S_0, \mathcal{F})$, where $\mathcal{F} = \{F_1, \ldots, F_k\}$, the Büchi automaton $A' = (\Sigma, S', \delta', S'_0, F')$ defined as follows accepts the same language as A.

- $S' = S \times \{1, \ldots, k\}$.
- $S'_0 = S_0 \times \{1\}$.
- δ' is defined by $(t, i) \in \delta'((s, j), a)$ if

$$t \in \delta(s, a) \text{ and } \begin{cases} i = j & \text{if } s \notin F_j, \\ i = (j \bmod k) + 1 & \text{if } s \in F_j. \end{cases}$$

- $F' = F_1 \times \{1\}$.

The idea of the construction is that the states of A' are the states of A marked by an integer in the range $[1, k]$. The mark is unchanged unless one goes through a state in F_j, where j is the current value of the mark. In that case the mark is incremented (reset to 1 if it is k). If one repeatedly cycles through all the marks, which is necessary for F' to be reached infinitely often, then all sets in $\mathcal{F}$ are visited infinitely often. Conversely, if it is possible to visit all sets in $\mathcal{F}$ infinitely often in A, it is possible to do so in the order F_1, F_2, $\ldots$ F_k and hence to infinitely often go through F' in A'.

Example 2. Figure 3 shows the Büchi automaton equivalent to the generalized Büchi automaton of Figure 2 whose acceptance condition is $\mathcal{F} = \{\{s_0\}, \{s_1\}\}$.

Nondeterministic[1] Büchi automata have many interesting properties. In particular they are closed under all Boolean operations as well as under projection.

[1] Deterministic Büchi automata are less powerful and do not enjoy the same properties (see for instance [15]).

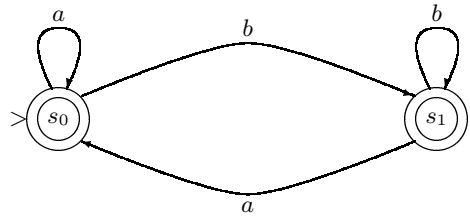


Fig. 2. A generalized Büchi automaton

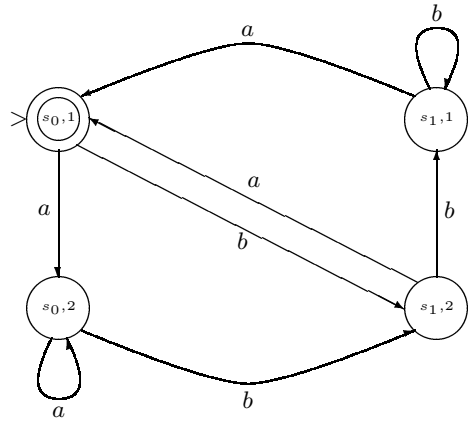


Fig. 3. From generalized Büchi to Büchi

Closure under union, projection are immediate given that we are dealing with nondeterministic automata; closure under intersection is obtained using a product construction similar to the one employed for finite-word automata. Closure under complementation is much more tricky and has been the subject of an extensive literature [1,14,12,9]. Checking that a (generalized) Büchi automaton is nonempty (accepts at least one word) can be done by computing its strongly connected components, and checking that there exists a reachable strongly connected component that has a non empty intersection with each set in $\mathcal{F}$.

4 From Temporal Logic to Automata

4.1 Problem Statement

We now consider the following problem: given an LTL formula φ built from a set of atomic propositions P, construct an automaton on infinite words over the alphabet 2^P that accepts exactly the infinite sequences satisfying φ.

To get an intuitive idea of what we are aiming at, let us first look at an example.

Example 3. Consider the formula $\Diamond p$. This formula describes the sequences over $\{\emptyset, \{p\}\}$ in which $\{p\}$ occurs at least once. These sequences are accepted by the automaton of Figure 4.

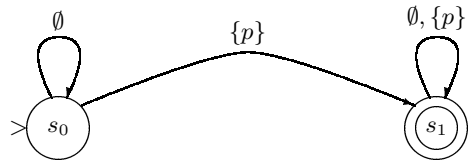


Fig. 4. An automaton for $\Diamond\, p$

4.2 The Closure of a Formula

In order to develop a procedure for building automata from LTL formulas, we first look at the problem of determining if a sequence $\sigma : \mathbf{N} \to 2^P$ satisfies a formula φ defined over the set of propositions P. This can, at least conceptually, be done by labeling the sequence with subformulas of φ in a way that respects LTL semantics. First, let us define the set of subformulas of a formula φ that are needed. This set is called the *closure* of φ ($cl(\varphi)$) and is defined as follows:

- $\varphi \in cl(\varphi)$,
- $\varphi_1 \wedge \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\varphi_1 \vee \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\bigcirc \varphi_1 \in cl(\varphi) \Rightarrow \varphi_1 \in cl(\varphi)$,
- $\varphi_1\, U\, \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\varphi_1\, \tilde{U}\, \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$.

Example 4.

$$cl(\Diamond\, \neg p) = cl(\mathbf{true}\, U\, \neg p) = \{\Diamond\, \neg p, \neg p, \mathbf{true}\}$$

4.3 Rules for Labeling Sequences

The next step is to define the set of rules that a valid *closure labeling* $\tau : \mathbf{N} \to 2^{cl(\varphi)}$ of a sequence $\sigma : \mathbf{N} \to 2^P$ has to satisfy. The validity criterion is that, if a formula $\varphi_1 \in cl(\varphi)$ labels a position i (i.e. $\varphi_1 \in \tau(i)$), then the sequence σ^i satisfies it ($\sigma^i \models \varphi_1$)[2]. For this to hold, our labeling rules have to mirror the semantic rules for LTL. A first set of rules deals with the purely propositional part of LTL.

Consider a closure labeling $\tau : \mathbf{N} \to 2^{cl(\varphi)}$ of a sequence $\sigma : \mathbf{N} \to 2^P$ for a formula φ defined over a set of atomic propositions P. For τ to be a valid labeling, it has to satisfy the following rules for every $i \geq 0$:

1. **false** $\notin \tau(i)$;
2. for $p \in P$, if $p \in \tau(i)$ then $p \in \sigma(i)$, and if $\neg p \in \tau(i)$ then $p \notin \sigma(i)$;

[2] Such a validly labeled structure is often called a Hintikka structure in the modal logic literature

3. if $\varphi_1 \wedge \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ and $\varphi_2 \in \tau(i)$;
4. if $\varphi_1 \vee \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ or $\varphi_2 \in \tau(i)$.

Note that the labeling rules are "if" rules and not "if and only if" rules. They give the requirements that a valid closure labeling *must* satisfy, but they do not require that labelings be maximal: there can be formulas of the closure that are satisfied at a given position, but that are not included in the label of that position.

Let us now turn to elements of the closure whose main operator is temporal. For the operator $\bigcirc$, the rule is quite immediate. We have that for all $i \geq 0$,

5. if $\bigcirc \varphi_1 \in \tau(i)$ then $\varphi_1 \in \tau(i+1)$.

For the U and $\tilde{U}$ operators, the semantic rules refer to a possibly infinite set of points of the sequence, which we would like to avoid in our labeling rules. Fortunately, this is mostly possible. Indeed, one can fairly easily show from the semantic rules that the following identities hold:

$$\varphi_1 \, U \, \varphi_2 \equiv (\varphi_2 \vee (\varphi_1 \wedge \bigcirc(\varphi_1 \, U \, \varphi_2)))$$
$$\varphi_1 \, \tilde{U} \, \varphi_2 \equiv (\varphi_2 \wedge (\varphi_1 \vee \bigcirc(\varphi_1 \, \tilde{U} \, \varphi_2))).$$

These identities then suggest the following labeling rules for all positions $i \geq 0$:

6. if $\varphi_1 \, U \, \varphi_2 \in \tau(i)$ then either $\varphi_2 \in \tau(i)$, or $\varphi_1 \in \tau(i)$ and $\varphi_1 \, U \, \varphi_2 \in \tau(i+1)$;
7. if $\varphi_1 \, \tilde{U} \, \varphi_2 \in \tau(i)$ then $\varphi_2 \in \tau(i)$, and either $\varphi_1 \in \tau(i)$ or $\varphi_1 \, \tilde{U} \, \varphi_2 \in \tau(i+1)$.

The rule for $\tilde{U}$ is sufficient to ensure that the labeling is valid. Unfortunately, the same is not true for the operator U. Indeed, rule 6 does not force the existence of a point at which φ_2 appears: such a point can be postponed forever. We thus need to add one more labeling rule, which unfortunately does not only refer to consecutive points. For every position $i \geq 0$, we must have that

8. if $\varphi_1 \, U \, \varphi_2 \in \tau(i)$ then there is a $j \geq i$ such that $\varphi_2 \in \tau(j)$.

As a hint at the requirement expressed by rule 8, a formula of the form $\varphi_1 \, U \, \varphi_2$ is often referred to as an *eventuality* since it requires that the formula φ_2 be *eventually* true. Rule 8 is then said to require that the eventualities are *fulfilled*.

We can now formalize the fact that the labeling rules we have given characterize the valid labelings. First we show that the labeling rules only allow valid labelings.

Lemma 2. *Consider a formula φ defined over a set of propositions P, a sequence $\sigma : \mathbf{N} \to 2^P$, and a closure labeling $\tau : \mathbf{N} \to 2^{cl(\varphi)}$ satisfying rules 1–8. For every formula $\varphi' \in cl(\varphi)$ and $i \geq 0$, one has that if $\varphi' \in \tau(i)$ then $\sigma^i \models \varphi'$.*

Proof. The proof proceeds by structural induction on the formulas of $cl(\varphi)$. Let us consider the most interesting case, which is that of a formula φ' of the form

$\varphi_1 U \varphi_2$. By rule 8, one has that there is a $j \geq i$ such that $\varphi_2 \in \tau(j)$ and, by inductive hypothesis, such that $\sigma^j \models \varphi_2$. Consider the smallest such j and a k such that $i \leq k < j$. Since $\varphi_1 U \varphi_2 \in \tau(i)$, and since for all $i \leq k' \leq k$, $\varphi_2 \notin \tau(k')$, rule 6 implies that $\varphi_1 U \varphi_2 \in \tau(k)$ and also that $\varphi_1 \in \tau(k)$. Hence, by inductive hypothesis, $\sigma^k \models \varphi_1$.

Next we need to establish that when a sequence satisfies a formula, a closure labeling satisfying rules 1–8 exists.

Lemma 3. *Consider a formula φ defined over a set of propositions P and a sequence $\sigma : \mathbf{N} \to 2^P$. If $\sigma \models \varphi$, there exists a closure labeling $\tau : \mathbf{N} \to 2^{cl(\varphi)}$ satisfying rules 1–8 and such that $\varphi \in \tau(0)$.*

Proof. Consider the closure labeling defined by $\varphi' \in \tau(i)$ iff $\sigma^i \models \varphi'$ for all $\varphi' \in cl(\varphi)$. Given that $\sigma \models \varphi$, one immediately has that $\varphi \in \tau(0)$. Furthermore, that fact that rules 1–8 are satisfied is a direct consequence of the semantics of LTL.

The following theorem is then a direct consequence of Lemmas 2 and 3.

Theorem 1. *Consider a formula φ defined over a set of propositions P and a sequence $\sigma : \mathbf{N} \to 2^P$. One then has that $\sigma \models \varphi$ iff there is a closure labeling $\tau : \mathbf{N} \to 2^{cl(\varphi)}$ of σ satisfying rules 1–8 and such that $\varphi \in \tau(0)$.*

4.4 Defining the Automaton

Given Theorem 1, the construction of an automaton accepting the sequences satisfying a formula φ is almost immediate. Indeed, remember that an automaton accepts an ω-sequence when this sequence can be labeled by states of the automaton, while satisfying the constraints imposed by the transition relation as well as by the initial and accepting state sets. The idea is simply to use $2^{cl(\varphi)}$ as state set and hence as set of possible labels. It then remains to express the required properties of the labeling by an appropriate definition of the structure of the automaton. We now show how this can be done.

Given a formula φ, a generalized Büchi automaton accepting exactly the sequences $\sigma : \mathbf{N} \to 2^P$ satisfying φ can be defined as follows. The automaton is $A_\varphi = (\Sigma, S, \delta, S_0, \mathcal{F})$ where

- $\Sigma = 2^P$,
- The set of states S is the set of possible labels, i.e. the subsets $\mathbf{s}$ of $2^{cl(\varphi)}$ that satisfy
  - **false** $\notin \mathbf{s}$;
  - if $\varphi_1 \wedge \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ and $\varphi_2 \in \mathbf{s}$;
  - if $\varphi_1 \vee \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ or $\varphi_2 \in \mathbf{s}$.

  The states (and hence possible labels) are thus the subsets of $2^{cl(\varphi)}$ that satisfy rules 1 as well as rules 3 and 4.

- The transition function δ checks that the propositional labeling matches the one in the sequence being considered (rule 2) and that the rules 5–7 for the temporal operators are satisfied. Thus, $\mathbf{t} \in \delta(\mathbf{s}, \mathbf{a})$ iff
  - For all $p \in P$, if $p \in \mathbf{s}$ then $p \in \mathbf{a}$.
  - For all $p \in P$, if $\neg p \in \mathbf{s}$ then $p \notin \mathbf{a}$.
  - if $\bigcirc \varphi_1 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{t}$.
  - if $\varphi_1 \, U \, \varphi_2 \in \mathbf{s}$ then either $\varphi_2 \in \mathbf{s}$, or $\varphi_1 \in \mathbf{s}$ and $\varphi_1 \, U \, \varphi_2 \in \mathbf{t}$.
  - if $\varphi_1 \, \tilde{U} \, \varphi_2 \in \mathbf{s}$ then $\varphi_2 \in \mathbf{s}$ and either $\varphi_1 \in \mathbf{s}$, or $\varphi_1 \, \tilde{U} \, \varphi_2 \in \mathbf{t}$.
- The set of initial states is defined in order to ensure that φ appears in the label of the first position of the sequence. We thus have that $S_0 = \{\mathbf{s} \in S \mid \varphi \in \mathbf{s}\}$.
- The acceptance condition $\mathcal{F}$ is used to impose rule 8 on the fulfillment of eventualities, but seeing how this can be done requires a slightly closer look at this requirement.

What needs to be imposed to satisfy rule 8 is that, for every eventuality formula $\varphi_1 \, U \, \varphi' \equiv e(\varphi') \in cl(\varphi)$, any state that contains that formula is followed by a state that contains φ'. The problem with the way this requirement is stated is that it requires "remembering" that a state in which $e(\varphi')$ occurs has been seen and hence extending the set of states of the automaton. Fortunately, this can be avoided.

Rule 6 (and hence the transition relation of the automaton) requires that if an eventuality $e(\varphi')$ appears, it keeps on appearing until the first state in which φ' appears. So, the only problematic situation would be one in which $e(\varphi')$ appears indefinitely without φ' ever appearing. So its is sufficient to require that the automaton goes infinitely often through a state in which both $e(\varphi')$ and φ' appear or in which $e(\varphi')$ does not appear, the latter case allowing for the eventuality no longer to be required after some point. The acceptance condition of the automaton is thus the following generalized Büchi condition.

- If the eventualities appearing in $cl(\varphi)$ are $e_1(\varphi_1), \ldots e_m(\varphi_m)$, $\mathcal{F} = \{\Phi_1, \ldots, \Phi_m\}$ where $\Phi_i = \{\mathbf{s} \in S \mid e_i, \varphi_i \in \mathbf{s} \vee e_i \notin \mathbf{s}\}$.

Given the way we have expressed the semantics of LTL in terms of labeling rules and given the semantics of Büchi automata, the correctness of the construction we have just given is essentially immediate.

Example 5. The automaton for $\Diamond p$ is given in Figure 5, where $\mathcal{F} = \{\{1, 3, 4\}\}$.

Note that in this example we have already applied two optimizations to the construction. First, we have identified states containing **true** with the states defined by an otherwise identical set of formulas. Second, we have omitted the transitions leaving from nodes 3 and 4 to nodes 1 and 2. It is intuitively obvious that these transitions are not needed but, in the next section, we will generalize these types of optimizations and justify them precisely.

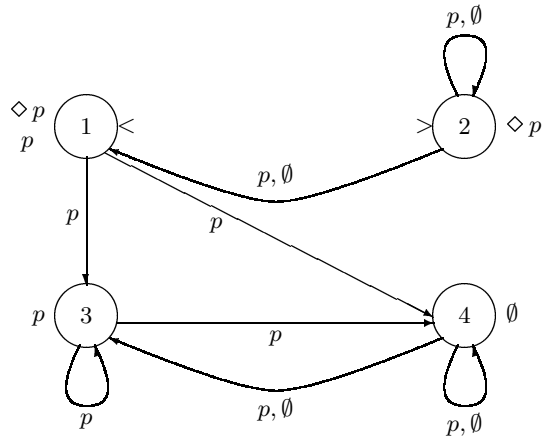


Fig. 5. The automaton constructed for $\Diamond p$

5 Improving the Construction

5.1 Omitting Redundant Transitions

The states of the automaton we build for a formula are subsets of the closure of that formula. The subset ordering thus naturally defines a partial order on the automaton states. Furthermore, given the way the transition relation is defined, any transition possible from a state $\mathbf{s}_1$ is also possible from any state $\mathbf{s}_2 \subset \mathbf{s}_1$. This seems to imply that if, from a given state, two transitions lead to states $\mathbf{s}_1$ and $\mathbf{s}_2$ such that $\mathbf{s}_2 \subset \mathbf{s}_1$, then it is sufficient to keep the transition leading to the state $\mathbf{s}_2$. Almost so. Indeed, the way the transition relation of the automaton is defined guarantees that if there is a computation of the automaton on a given word from $\mathbf{s}_1$, there is also one from $\mathbf{s}_2$. The problem is with accepting states: if $\mathbf{s}_1$ contains an eventuality formula $e(\varphi')$ as well as its argument φ', but that $\mathbf{s}_2$ only contains $e(\varphi')$, $\mathbf{s}_2$ might be outside an accepting set in which $\mathbf{s}_1$ is included. The simplification rule we will use is thus the following.

Omit transitions. Assume that from a state $\mathbf{s}$ two identically labeled transitions lead to states $\mathbf{s}_1$ and $\mathbf{s}_2$ such that $\mathbf{s}_2 \subset \mathbf{s}_1$ and such that, for all eventuality formulas $e(\varphi') \in \mathbf{s}_1$, if $e(\varphi') \in \mathbf{s}_2$ and $\varphi' \in \mathbf{s}_1$ then also $\varphi' \in \mathbf{s}_2$. The transition from $\mathbf{s}$ to $\mathbf{s}_1$ can then be omitted.

Example 6. Applying the **omit transitions** rule to the automaton of Figure 5 and eliminating unreachable states, one obtains the automaton of Figure 6.

To see that the **omit transitions** rule is sound, we establish that for every state of the automaton, the language accepted from that state after applying the **omit transitions** rule is unchanged. First notice that we are removing transitions. So, after applying the rule, the language accepted cannot contain more

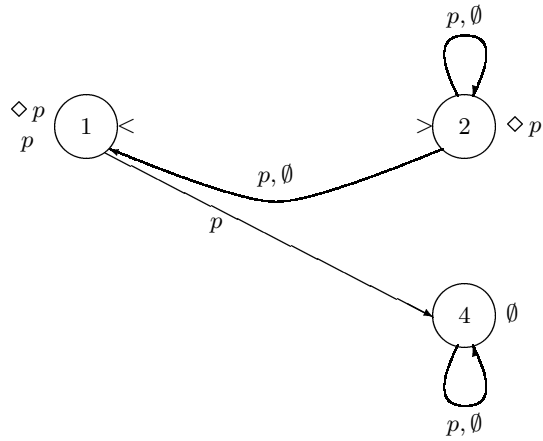


Fig. 6. A simplified automaton for $\Diamond p$

words. We show that it also cannot contain less words. Assume that there exists an accepting computation from a state $\mathbf{s}$ before applying the **omit transitions** rule. Such a computation still exists after applying the rule. Indeed, if the computation from $\mathbf{s}$ starts with an omitted transition leading to a state $\mathbf{s}_1$, there remains an identically labeled transition to a state $\mathbf{s}_2 \subset \mathbf{s}_1$ that is accepting whenever $\mathbf{s}_1$ is accepting. Now, since $\mathbf{s}_2 \subset \mathbf{s}_1$, before the transition omission procedure, all transitions possible from $\mathbf{s}_1$ are also possible from $\mathbf{s}_2$, so there also is an accepting computation from $\mathbf{s}_2$. Of course, some transitions from $\mathbf{s}_2$, may also have been omitted, but the same argument can be repeated for the computation starting at $\mathbf{s}_2$. By induction, one can then conclude the existence of the required accepting computation in the simplified automaton.

5.2 Building the Automaton by Need

The most obviously wasteful aspect of the construction we have shown is that it defines the set of states to be all subsets of the closure that satisfy the rules 1, 3, and 4. Indeed, many of these states might not be reachable from initial states, especially if the **omit transitions** simplification rule is applied. To avoid this, we are going to construct the states of the automaton as needed, starting with the initial states and adding the states that must appear as targets of transitions.

Preparing to do this, notice that all the rules embodied in the transitions of the automaton require that, if some formula of the closure occurs in the current state, then some other formula also occurs in the current or next state. Furthermore, the **omit transitions** simplification allows us to only consider the minimal states satisfying these conditions. This leads us to defining an operation that adds to a subset of the closure all formulas that *must* be true in the current and in the immediately next state. For ease of definition, we define this operation (*saturate*) not on subsets of the closure, but on sets of subsets of the closure.

Let $\mathbf{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\} \subseteq 2^{cl(\varphi)}$, then we define *saturate* $(\mathbf{Q})$ as follows.

1. Repeat until stabilization: for each $\mathbf{q}_i \in \mathbf{Q}$,
   (a) If $\varphi_1 \wedge \varphi_2 \in \mathbf{q}_i$, then
   $\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_1, \varphi_2\}\}$;
   (b) If $\varphi_1 \vee \varphi_2 \in \mathbf{q}_i$, then
   $\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_1\}\} \cup \{\mathbf{q}_i \cup \{\varphi_2\}\}$;
   (c) If $\varphi_1 \, U \, \varphi_2 \in \mathbf{q}_i$, then
   $\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_2\}\} \cup \{\mathbf{q}_i \cup \{\varphi_1, \bigcirc(\varphi_1 \, U \, \varphi_2)\}\}$;
   (d) If $\varphi_1 \, \tilde{U} \, \varphi_2 \in \mathbf{q}_i$, then
   $\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_1, \varphi_2\}\} \cup \{\mathbf{q}_i \cup \{\varphi_2, \bigcirc(\varphi_1 \, \tilde{U} \, \varphi_2)\}\}$
2. Remove all $\mathbf{q}_i \in \mathbf{Q}$ such that $\mathbf{false} \in \mathbf{q}_i$

If the operation *saturate* is applied to a singleton $\mathbf{q}$, then the result is a set of sets of formulas[3] that represent possible ways of satisfying the requirements expressed by the formulas in $\mathbf{q}$. Among such sets of formulas, we will be especially interested in the propositional formulas and in the formulas having $\bigcirc$ as their main connective, which constrain the next state. We thus define the following filters on a set of LTL formulas $\mathbf{q}$:

1. $X(\mathbf{q}) = \{\varphi_i \mid \bigcirc \varphi_i \in \mathbf{q}\}$ (the "next" formulas in $\mathbf{q}$ with their $\bigcirc$ operator stripped),
2. $P(\mathbf{q}) = \{p_i \mid p_i \in \mathbf{q} \wedge p_i \in P\}$ (the atomic propositions in $\mathbf{q}$),
3. $nP(\mathbf{q}) = \{p_i \mid \neg p_i \in \mathbf{q} \wedge p_i \in P\}$ (the negated atomic propositions in $\mathbf{q}$).

We are now ready to give a "by need" algorithm for generating the automaton. For a formula φ, the algorithm generates an automaton $A_\varphi = (\Sigma, S, \delta, S_0, \mathcal{F})$. The alphabet and accepting condition are defined as before. The states and transitions are progressively generated by the algorithm. For ease of notation, we will represent the transition function δ as a set of triples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ where $\mathbf{s}, \mathbf{s}' \in S$ and $\mathbf{a} \in \Sigma$.

The algorithm works with a list of unprocessed states for which successors still have to be generated. For these unprocessed states, additional information in the form of "next requirements", i.e. the formulas that have to be true in all the immediate successors of the unprocessed state, is maintained. Unprocessed states thus take the form of a pair of sets of formulas $(\mathbf{s}, \mathbf{x})$, where $\mathbf{s}$ is the state and $\mathbf{x}$ the "next requirements". The unprocessed states are stored in a list *unp*. The automaton building procedure is then the following.

[3] These sets are not strictly subsets of the closure since rules 1c and 1d can generate formulas that are elements of the closure preceded by the $\bigcirc$ operator.

build-auto (φ)
1. $S := \emptyset$; $\delta := \emptyset$; $S_0 := \{\mathbf{q} \cap cl(\varphi) \mid \mathbf{q} \in saturate(\{\{\varphi\}\})\}$
2. $unp := \{(\mathbf{q} \cap cl(\varphi), X(\mathbf{q})) \mid \mathbf{q} \in saturate(\{\{\varphi\}\})\}$
3. **while** $unp \neq \emptyset$ **do**
   Choose and remove $(\mathbf{s}, \mathbf{x})$ from unp;
   $S := S \cup \{\mathbf{s}\}$;
   For each $\mathbf{q} \in saturate(\mathbf{x})$ **do**
   For each $\mathbf{a} \in \Sigma$ **such that** $P(\mathbf{q}) \subseteq \mathbf{a} \wedge nP(\mathbf{q}) \cap \mathbf{a} = \emptyset$
   $\delta := \delta \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{q} \cap cl(\varphi))\}$
   if $(\mathbf{q} \cap cl(\varphi), X(\mathbf{q})) \notin unp \wedge \mathbf{q} \cap cl(\varphi) \notin S$
   then $unp := unp \cup \{(\mathbf{q} \cap cl(\varphi), X(\mathbf{q}))\}$

Note that states are restricted to be subsets of the closure of the initial formula. This is not essential, but guarantees that the automaton built by the procedure above is a subset of the one built by the abstract construction of Section 4. Thus, it only accepts words also accepted by this automaton. That it accepts all words accepted by this automaton is a direct consequence of the fact that the *saturate* operation generates minimal sets (it only includes formulas that *must* be present) and of the argument used to justify the **omit transitions** simplification rule. The only somewhat delicate point concerns the special requirement on eventuality formulas imposed by the **omit transitions** rule. But, starting with a set containing an eventuality $e(\varphi')$, the *saturate* procedure always generates a set containing φ', the presence of suitable accepting states is thus guaranteed.

Example 7. Applying the *build-auto* algorithm to $\Diamond p$, will produce the automaton in the following stages.

1. First, the initial states $\{\Diamond p, p\}$ and $\{\Diamond p\}$ are produced, with *unp* set to $\{(\{\Diamond p, p\}, \emptyset), (\{\Diamond p\}, \{\Diamond p\})\}$.
2. $(\{\Diamond p\}, \{\Diamond p\})$ is removed from *unp* and transitions labeled by p and $\emptyset$ are created from the states$\{\Diamond p\}$ to itself and to the state $\{\Diamond p, p\}$.
3. $(\{\Diamond p, p\}, \emptyset)$ is removed from *unp* and a transition labeled p from the $\{\Diamond p, p\}$ state to the state $\emptyset$ is added; $(\emptyset, \emptyset)$ is added to *unp*.
4. $(\emptyset, \emptyset)$ is removed from *unp* and transitions labeled by p and $\emptyset$ are created from the state $\emptyset$ to itself.

5.3 Identifying Equivalent States

One rather obvious limit of the "by need" procedure we have outlined, is that it only identifies states that are syntactically identical, i.e. consist of exactly the same set of formulas. A further reduction in the size of the automaton can thus be obtained by attempting to identify states that are semantically identical, i.e. that define identical sets of temporal sequences. Of course, deciding semantical equivalence in general is as hard as building an automaton from a temporal logic formula, and cannot be usefully used during the construction. However, one can identify some common semantical equivalences that can substantially reduce the

size of the automaton. The following have, for instance, been used successfully [6,4,13].

- $\{\varphi_1, \varphi_2, \varphi_1 \wedge \varphi_2\} \Leftrightarrow \{\varphi_1, \varphi_2\}$
- $\{\varphi_1, \varphi_1 \vee \varphi_2\} \Leftrightarrow \{\varphi_1\}$
- $\{\varphi_2, \varphi_1 \vee \varphi_2\} \Leftrightarrow \{\varphi_2\}$
- $\{\varphi_2, \varphi_1 \, U \, \varphi_2\} \Leftrightarrow \{\varphi_2\}$

The only caveat while using such semantical equivalences, is that they might change the definition of accepting states. One easy way around this is to disallow using such semantical equivalences involving formulas that are the argument of eventualities.

5.4 Further Improvements

There are a number of further improvements that can be made to the construction of an automaton from a temporal logic formula. We briefly describe a few.

Simplifying the formula. Before applying the construction, it can be useful to rewrite the formula using some equivalence preserving transformation. For instance, one can use $\bigcirc \Box \Diamond \varphi \equiv \Box \Diamond \varphi$ to remove a "next" operator from a formula.

Early detection of inconsistencies. With the procedure we have outlined so far, inconsistencies are only detected after being propagated to the propositional level. Clearly, if a state contains both the formulas φ_1 and $\neg\varphi_1$ it is inconsistent and no transitions need leave that state.

Moving propositions from states to transitions. Propositional constraints are included in the states and uniformly applied to all transitions leaving a given state. In many cases this is wasteful and leads to excessively nondeterministic automata. An alternative is to let the choice of next state be determined by the propositions that are actually received as input. This allows propositional requirements to be removed from, and moved exclusively to, the transitions. However, special attention must be paid to the case in which propositions are the argument of eventualities. Indeed, removing them from states will then impact the definition of acceptance conditions.

Simplifying the acceptance condition. It is not uncommon for the structure of the strongly connected components of the automaton to allow a simplifying of the acceptance condition. This can lead to improved efficiency in the use of the automaton.

6 Conclusions

The intrinsically exponential complexity of building automata from temporal logic formulas, has long been seen as a limiting factor to the use of linear-time model checking. However, this conclusion ignores two important facts. First the

formulas used in specifications are almost always very short. Second, the worst-case complexity bounds on building automata from temporal logic formulas are just that: *worst-case* bounds. The work surveyed in this paper shows that in the very large majority of cases, it is possible to build automata of very reasonable size for any temporal formula one would care to write in a specification. Of course, it is possible to produce pathological cases. For instance, using n propositions, one can polynomially encode an n bit counter in temporal logic, the corresponding automaton necessarily being exponential in n.

However, if one fixes the number of propositions, it is much less obvious to build a family of formulas for which the size of the corresponding automaton unavoidably exhibits exponential growth. In the opinion of this author, it is even highly unlikely that one would come upon such formulas when specifying program properties. The doubtful reader is invited to attempt to construct such a family of formulas.

So, in conclusion, it can be said that the work on improving the practical behavior of algorithms for generating automata from temporal logic formulas [6], [4], [13] has been successful to the point of showing that the inherently exponential nature of the problem is of little practical significance. This can be viewed as meaning that the lower bound proofs rely on using the expressive power of temporal logic in a way that is too unnatural to occur in many applications.

References

1. J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.
2. Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, California, June 1994. Springer-Verlag.
3. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.
4. M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Computer-Aided Verification, Proc. 11th Int. Conference*, volume 1633, pages 249–260, July 1999.
5. E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.
6. Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15th Work. Protocol Specification, Testing, and Verification*, Warsaw, June 1995. North-Holland.
7. G. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International Editions, 1991.
8. Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997. Special Issue: Formal Methods in Software Practice.

9. O. Kupferman and M. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.
10. Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, January 1984.
11. J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int'l Symp. on Programming*, volume 137, pages 337–351. Springer-Verlag, Lecture Notes in Computer Science, 1981.
12. S. Safra. On the complexity of omega-automata. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, White Plains, October 1988.
13. F. Somenzi and R. Bloem. Efficient büchi automata from ltl formulae. In *Computer-Aided Verification, Proc. 12th Int. Conference*, volume 1633, pages 247–263, 2000.
14. A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
15. Wolfgang Thomas. Automata on infinite objects. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science – Volume B: Formal Models and Semantics*, chapter 4, pages 133–191. Elsevier, Amsterdam, 1990.
16. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.
17. Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.
18. Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proc. 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, Tucson, 1983.

Exploiting Structure in Solution: Decomposing Compositional Models

Jane Hillston

Division of Informatics, University of Edinburgh,
jeh@dcs.ed.ac.uk

Abstract. Since their introduction in the early 1990s, compositionality has been reported as one of the major attractions of stochastic process algebras. The benefits that compositionality provides for model construction are readily apparent and have been demonstrated in numerous case studies. Early research on the compositionality of the languages focused on how the inherent structure could be used, in conjunction with equivalence relations, for model simplification and aggregation. In this chapter we consider how far we have been able to take advantage of compositionality when it comes to solving the Markov process underlying a Markovian process algebra model.

1 Introduction

At the time of the introduction of stochastic process algebras (SPA) [30] there was already a plethora of techniques for constructing performance models so the introduction of another one could have been deemed unnecessary if it were not for the fact that SPA offered something new—formally defined compositionality. Queueing networks, which have been widely used for performance modelling for more than thirty years, have an inherent compositionality but this is implicit and informal. Stochastic extensions of Petri nets have a semantic model but, in general, no clear compositional structure. In the process algebra the compositionality is explicit—provided by the combinators of the language—and formal—supported by the semantics and equivalence relations of the language.

It was immediately clear that having this explicit structure within models offers benefits for model construction:

- when a system consists of interacting components, the components, and the interaction, can each be modelled separately;
- models have a clear structure and are easy to understand;
- models can be constructed systematically, by either elaboration or refinement;
- the possibility of maintaining a library of model components, supporting model reusability, is introduced.

Many case studies demonstrating these and other benefits have appeared in the literature [33,35,55,23,41,1,21], and examples have been given in earlier chapters.

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 278–314, 2001.

However, almost as quickly, it became clear that SPA models are prone to problems of state space explosion: making it easy for the modeller to represent systems in detail, coupled with the inherent complexity of the systems of interest, inevitably leads to models which are extremely large; in many cases, intractably so. In particular, coupled with the abstraction provided by the hiding combinator, compositionality allows a modeller to represent components of the system in detail, model their interaction in appropriate ways, and then abstract from the internal details of the combined component. This is a good technique for capturing the behaviour of systems. But note that although abstraction reduces the observability of actions, and in some languages reduces the measures that can be made on the model, it does not generally eliminate the internal states. Thus this attractive "feature" of SPA actually exacerbates the state space explosion problem.

In the context of Markovian process algebras (MPA), tackling this problem has been a major motivation of much research. Of course, there are two state spaces which may be considered—the state space of the MPA model, which is generated in the labelled transition system via the operational semantics; and the state space of the underlying Markov process, the model to be solved (see Figure 1). Using the most straightforward procedure for generating the underlying Markov process, there is an isomorphism between the two. We could try to attack the state space explosion problem at either level; indeed, there exists a substantial body of literature considering the problem of state space explosion at the level of the Markov process via a variety of techniques. But in order to benefit fully from the process algebra apparatus we choose to work at the level of the MPA.

Initial efforts to combat the state space explosion problem concentrated on model manipulation techniques—model simplification and model aggregation. These techniques aim to *improve*, from the perspective of solution, the underlying Markov process, via manipulations of the state space at the MPA level. The simplest form of improvement is a reduction in the number of states. However, there are other possibilities, as we will discuss at the end of the chapter. At the core of these techniques are equivalence relations, but compositionality also has an important role to play, as we will discuss in Section 3.

Unfortunately model manipulation alone still leaves us with a Markov process which must be solved numerically as a single entity. This leads to the inevitable question of how the compositional structure within MPA models relates to the various decomposed solution techniques which can be applied to Markov processes. We survey MPA work in this area in some detail in Section 5. Within the possible decomposed solution techniques we will focus primarily on *product form approaches*. These offer *exact solution* in the sense that the steady state distribution of the original model can be recovered, when the decomposition satisfies certain (stringent) criteria. Nevertheless we will also discuss some approximative decomposed solution techniques, which generally have wider applicability, in Section 6.

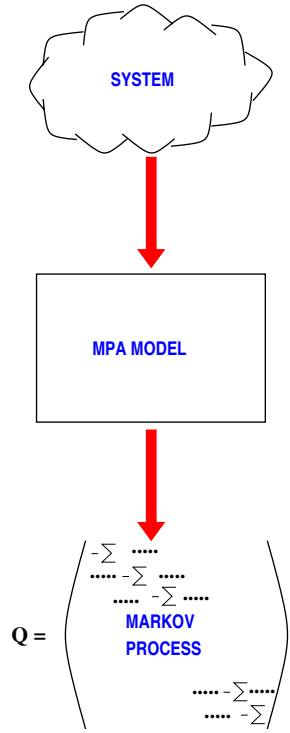


Fig. 1. Schematic view of MPA modelling methodology

In both approaches the work often exhibits common elements, although different researchers concentrate on different aspects of this general framework. In order for a decomposed solution technique to be fully developed and exploited in a tool the following steps have to be established:

- *Characterisation of structures within the MPA model which correspond to decomposable structure in the underlying Markov process.* The aim of this work is to identify those MPA models which have a structure which is amenable to decomposed solution. In some cases this is fairly informal; in others the aim has been to establish syntactic rules which may be applied automatically, meaning that a tool may analyse a given model and decide whether it falls within the decomposable class or not.
- *Development of revised algorithms to generate the decomposed Markov process (usually as a set of Markov processes) from the MPA model.* Identifying the set of Markov processes susceptible to efficient solution is only the first step. In the "standard" solution algorithm there is a straightforward mapping from the semantic model to the state space of the underlying Markov process. If this process is to be decomposed, more sophisticated Markov process generation algorithms may be needed. Once the composite Markov pro-

cesses are formed, in some cases known solution algorithms can be applied; in some existing techniques can be modified; in others, new approaches to decomposed solution have been suggested by the process algebra structure. In the latter case, the new solution algorithms must also be developed.
– *Implementation of related algorithms.* Prototype tools and implementations of the new techniques, for characterisation, Markov process generation and decomposed solution, allow them to be tested in practice, and eventually, put to practical use.

Each of the decomposed solution techniques is limited in its scope of application if corresponding results and performance measures are not to be too seriously compromised. However, the real opportunity to exploit the benefits of decomposed solution is likely to arise when the decomposition structure is used as a target for model manipulation. In this case a given model is manipulated into a form which is more amenable to efficient solution, not by reducing the number of states *per se*, but by manipulating its structure into a form suitable for a decomposed solution. This is a major area of current work and will be discussed in more detail in Section 7.

As mentioned earlier, the focus of this chapter will be a class of decomposed solution techniques which result in exact product form solution of the steady state distribution. All work in this area for MPA has been carried out in the context of the MPA language PEPA (Performance Evaluation Process Algebra). Consequently, in the following section, we give a brief introduction to PEPA; the interested reader is referred to [33] for more details.

2 PEPA

The basic elements of PEPA are *components* and *activities*, corresponding to *states* and *transitions* in the underlying Markov process. Each activity has an *action type* (or simply *type*). Activities which are private to the component in which they occur are represented by the distinguished action type, τ. The duration of each activity is represented by the parameter of the associated exponential distribution: the *activity rate* (or simply *rate*) of the activity. This parameter may be any positive real number, or the distinguished symbol $\top$ (read as *unspecified*). Thus each activity, a, is a pair (α, r) where α is the action type and r is the activity rate. We assume that there is a countable set of components, which we denote $\mathcal{C}$, and a countable set, $\mathcal{A}$, of all possible action types. We denote by $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^+$, the set of activities, where $\mathbb{R}^+$ is the set of positive real numbers together with the symbol $\top$.

2.1 Syntax and Informal Semantics

PEPA provides a small set of combinators. These allow expressions, or terms, to be constructed defining the behaviour of components, via the activities they undertake and the interactions between them. The combinators, together with

their names and interpretations, are presented informally below. The essential idea of these combinators will be familiar from earlier chapters, but they are included here for clarity.

Prefix: $(\alpha, r).P$ Prefix is the basic mechanism by which the behaviours of components are constructed. The component carries out activity (α, r) and subsequently behaves as component P.

Choice: $P + Q$ The component represents a system which may behave either as component P or as Q: all the current activities of both components are enabled. The first activity to complete, determined by a *race condition*, distinguishes one component, the other is discarded.

Cooperation: $P \underset{L}{\bowtie} Q$ The components proceed independently with any activities whose types do not occur in the *cooperation set* L (*individual activities*). However, activities with action types in the set L require the simultaneous involvement of both components (*shared activities*). These activities are only enabled in $P \underset{L}{\bowtie} Q$ when they are enabled in both P and Q. The cooperation combinator associates to the left but brackets may also be used to clarify the meaning. When the set L is empty, we use the more concise notation $P \parallel Q$ to represent $P \underset{\emptyset}{\bowtie} Q$.

The published stochastic process algebras differ on how the rate of shared activities are defined [31]. In PEPA the shared activity occurs at the rate of the slowest participant (see Appendix A for details). If an activity has an unspecified rate in a component, the component is *passive* with respect to that action type. A model which contains a passive activity without a partner for cooperation is considered to be *incomplete*.

Hiding: P/L The component behaves as P except that any activities of types within the set L are *hidden*, i.e. such an activity exhibits the unknown type τ and the activity can be regarded as an internal delay by the component. Such an activity cannot be carried out in cooperation with any other component: the original action type of a hidden activity is no longer externally accessible, to an observer or to another component; the duration is unaffected.

Constant: $A \stackrel{\text{def}}{=} P$ associates the constant A with the behaviour of the component P. This is how we assign names to components (behaviours). There is no explicit recursion operator but components of infinite behaviour may be readily described using sets of mutually recursive defining equations.

The action types which the component P may next engage in are the *current action types* of P, a set denoted $\mathcal{A}(P)$. This set is defined inductively over the syntactic constructs of the language (see [33] for a formal definition). For example, $\mathcal{A}(P+Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$. The activities which the component P may next engage in are the *current activities* of P, a multiset denoted $\mathcal{A}ct(P)$. When the system is behaving as component P these are the activities which are enabled. Note that the dynamic behaviour of a component depends on the number of instances of each enabled activity and therefore we consider *multisets* of activities as opposed to *sets* of action types.

Example: Simple processing system as cooperating components Consider a simple system in which a process repeatedly carries out some task. In order to complete

its task the process needs the cooperation of a subsidiary process for part, but not all, of the time. Thus the task can be regarded as being in two stages. The subsidiary process meanwhile has only two activities, it is available for use except for a short period after use while it is reset. We model the process and the subsidiary process as two separate components: *Process* and *Sub* respectively. The process will undertake two activities consecutively: *use* with some rate r_1, in cooperation with the subsidiary process, and *task* at rate r_2, representing the remainder of its processing task. Similarly the subsidiary process will engage in two activities consecutively: *use*, at rate r_3 and *reset*, at rate r_4.

$$\begin{array}{ll} Process \stackrel{\text{def}}{=} (use, r_1).Process' & Sub \stackrel{\text{def}}{=} (use, r_3).Sub' \\ Process' \stackrel{\text{def}}{=} (task, r_2).Process & Sub' \stackrel{\text{def}}{=} (reset, r_4).Sub \end{array}$$

$$System \stackrel{\text{def}}{=} Process \underset{\{use\}}{\bowtie} Sub$$

Note that we can easily extend the model to represent a system with two primary processes, independent of each other but competing for the use of the subsidiary process: $(Process \parallel Process) \underset{\{use\}}{\bowtie} Sub$.

2.2 Execution Strategy

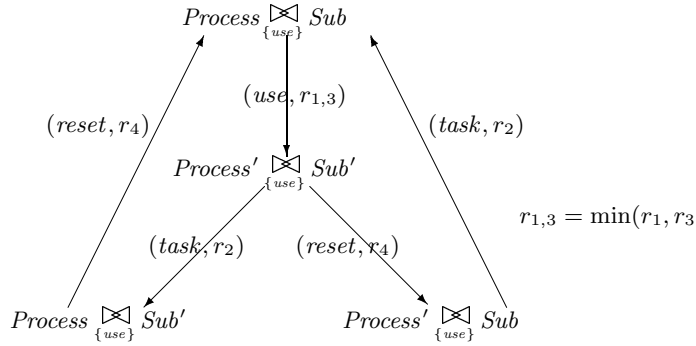
A *race condition* governs the dynamic behaviour of a model whenever more than one activity is enabled. This has the effect of replacing the non-deterministic branching of classical process algebra with probabilistic branching. The probability that a particular activity completes is given by the ratio of the activity rate to the sum of the activity rates of all the enabled activities. Any other activities which were simultaneously enabled will be *interrupted* or *aborted*. The memoryless property of the exponential distribution makes it unnecessary to record the remaining lifetime in either case.

2.3 Operational Semantics and the Underlying CTMC

The semantics of PEPA, presented in the structured operational semantics style, are given in Appendix A. The underlying transition system also characterises the Markov process represented by the model. PEPA is the labelled *multi*-transition system $(\mathcal{C}, \mathcal{A}ct, \{\!| \xrightarrow{(\alpha,r)} | \ (\alpha, r) \in \mathcal{A}ct |\!\})$ where $\mathcal{C}$ is the set of components, $\mathcal{A}ct$ is the set of activities and the multi-relation $\xrightarrow{(\alpha,r)}$ is given by the rules in Appendix A.

The *derivation graph* is a graph in which syntactic terms form the nodes, and arcs represent the possible transitions between them: the operational rules define the form of this graph. Since $\xrightarrow{(\alpha,r)}$ is a multi-relation, the graph is a multigraph. This derivation graph describes the possible behaviour of any PEPA component and provides a useful way to reason about a model. It is also the basis of the construction of the underlying Markov process.

Example: Simple processing system – derivation graph



Definition 1 (Derivatives). *If* $P \xrightarrow{(\alpha,r)} P'$*, then* P' *is a* (one-step) derivative *of* P*. In general,* P' *is a* derivative *of* P *if* $P \xrightarrow{(\alpha_1,r_1)} \cdots \xrightarrow{(\alpha_n,r_n)} P'$*.*

These derivatives are the states of the labelled multi-transition system (and of the underlying Markov process). The set of derivatives which can evolve from a component is defined recursively.

Definition 2 (Derivative Set). *The* derivative set *of a PEPA component* C *is denoted* $ds(C)$ *and defined as the smallest set of components such that*

- *if* $C \stackrel{\text{def}}{=} C_0$ *then* $C_0 \in ds(C)$*;*
- *if* $C_i \in ds(C)$ *and there exists* $a \in \mathcal{A}ct(C_i)$ *such that* $C_i \xrightarrow{a} C_j$ *then* $C_j \in ds(C)$*.*

Thus the derivative set is the set of components which capture all the reachable states of the system. These form the nodes of the derivation graph.

Definition 3 (Derivation Graph). *Given a PEPA component* C *and its derivative set* $ds(C)$*, the* derivation graph $\mathcal{D}(C)$ *is the labelled directed multi-graph, whose set of nodes is* $ds(C)$*, and whose multiset of arcs,* A*, is defined as follows:*

- *The elements of* A *are taken from the set* $ds(C) \times ds(C) \times \mathcal{A}ct$*;*
- $\langle C_i, C_j, a \rangle$ *occurs in* A *with the same multiplicity as the number of distinct inference trees which imply* $C_i \xrightarrow{a} C_j$*.*

The initial component C_0*, where* $C \stackrel{\text{def}}{=} C_0$*, forms the initial node of the graph.*

To form the underlying Markov process a state is associated with each node of the derivation graph, and the transitions between states are derived from the arcs of the graph. This use of the derivation graph is analogous to the use of the reachability graph in stochastic extensions of Petri nets such as SPNs [52] and has been discussed in an earlier chapter. We assume that the model is finite so that the number of nodes in the derivation graph is finite.

The *transition rate* between two components C_i and C_j, denoted $q(C_i, C_j)$, is the sum of the activity rates labelling arcs connecting node C_i to node C_j in the derivation graph, i.e. $q(C_i, C_j) = \sum_{a \in \mathcal{A}ct(C_i|C_j)} r_a$ where $\mathcal{A}ct(C_i|C_j)$ is defined to be $\{\!|a \in \mathcal{A}ct(C_i) \mid C_i \xrightarrow{a} C_j|\!\}$. Typically this multiset will only contain one element. If C_j is not a one-step derivative of C_i, then $q(C_i, C_j) = 0$. The $q(C_i, C_j)$, or q_{ij}, are the off-diagonal elements of the infinitesimal generator matrix of the Markov process, $\mathbf{Q}$. Diagonal elements are formed as the negative sum of the non-diagonal elements of each row.

2.4 Cyclic PEPA

When aiming for a decomposed solution which may rely on the numerical solution of individual components within the model, it is important to ensure that these components, as well as the model itself, are finite and ergodic. Necessary (but not sufficient) conditions for the ergodicity of the Markov process in terms of the structure of the PEPA model have been identified and can be readily checked [33,24]. These conditions imply that the model must be a *cyclic* PEPA component.

Definition 4 (Cyclic Components). *A PEPA component is* cyclic*, or* irreducible*, if it is a derivative of all the components in its derivative set.*

$$C \in ds(C_i) \text{ for all } i \text{ such that } C_i \in ds(C)$$

A cyclic component is one in which behaviour may always be repeated—however the model evolves from this component it will always eventually return to this component and this set of behaviours. In particular this means that for every choice, whichever one-step derivative is chosen the model must eventually return to the point where the choice can be made again, possibly with a different outcome. If we consider the layering imposed on a component by cooperation combinators, this implies that choice combinators may only be introduced at the lowest level of a cyclic component since syntactic terms are associated with states. In other words, a component which involves a choice combinator may subsequently be used in a cooperation, but a component involving a cooperation may not be subsequently used in a choice.

This leads us to formally define the syntax of PEPA expressions in terms of *sequential components* S and *model components* P:

$$P ::= S \mid P \underset{L}{\bowtie} P \mid P/L$$
$$S ::= (\alpha, r).S \mid S + S \mid A$$

For the remainder of the chapter we will assume that all the models which we consider are cyclic.

As stated earlier there is a strong relationship between cyclic PEPA components and irreducibility in the underlying Markov processes. This is formalised in the following theorem.

Theorem 1. *The Markov process underlying a PEPA model is irreducible, and therefore ergodic, if, and only if, the initial component of the model is cyclic.*

As explained in the previous subsection the "states" of a PEPA model as it evolves are the syntactic terms, or derivatives, which the model will go through. For a model component the sequential components it consists of will be apparent in every derivative of the model[1], i.e. the sequential components in the model, and the cooperation sets in operation between them, will remain static throughout the evolution of the model. Only the particular derivatives exhibited by each of the sequential components may change. This suggests a compact representation of any particular state.

Definition 5 (State Vector). *Let P be a model component comprising sequential components $S_1, S_2, \ldots S_K$. Then a* state vector *of the model component P as derivative P_i is the vector $(S_{1_i}, S_{2_i}, \ldots, S_{K_i})_P$ where $S_{k_i}, 1 \leq k \leq K$ is the current derivative of S_k in P_i.*

This can be regarded as analogous to the state representation of a queueing network which consists of a vector $(n_1, n_2, \ldots, n_K)$, where n_i denotes the number of customers currently at queue i. The subscript P is required since knowledge of the static structure of P must be retained in order to reason about the model's behaviour. However, it will be omitted when it is clear from the context which P is intended.

Definition 6 (Redundancy (within the state vector representation)). *A sequential component S_k is* redundant within the state vector representation *of a model component P if S_k is a sequential component of P and for all derivatives $P_i \in ds(P)$ given the current derivatives of the other sequential components $S_{j_i}, j \neq k$, the current derivative of S_k, S_{k_i}, can be inferred.*

If a sequential component is shown to be redundant within the state vector, a *reduced state vector* may be formed in which the derivatives of this component have been eliminated.

3 Harnessing Compositionality

As outlined in Section 1, initial attempts to exploit the compositionality of MPA languages focused on model manipulation—improving the model in some sense before the underlying Markov process is generated and solved.

There have been two principal approaches to model manipulation:

model simplification: Here an equivalence relation is used to establish behavioural or observational equivalence *between models*. The aim is to replace one model by an equivalent one which is more desirable from a solution point

[1] However they will not necessarily all change with every transition of the derivation graph.

of view. Once the desirable model has replaced the original, the underlying Markov process is generated as usual, associating one state with each node in the labelled transition system generated by the semantics. Equivalence relations which have been used in this way are *weak isomorphism* in PEPA [33,14], *Markovian bisimulation* and *weak bisimulation* in TIPP [47].

model aggregation: Here an equivalence relation is used to establish equivalence *between states within a model*. The aim is to use an alternative mapping from the labelled transition system, given by the semantics of the model, to the underlying Markov process. The equivalence relation is used to partition the nodes of the labelled transition system into equivalence classes. Then, instead of the usual one-to-one correspondence between nodes and states, one state in the underlying Markov process is associated with each equivalence class of nodes. The equivalence relation which has been used in this way is variously called *strong equivalence* (PEPA) [33], *Markovian bisimulation* (TIPP) [29], and *extended Markovian bisimulation equivalence* (EMPA) [4].

In model aggregation the introduction of equivalence classes will generally reduce the number of states in the underlying Markov process and will certainly never increase it. Thus the resulting Markov process is more amenable to solution because its size has been reduced. Similarly, in model simplification reducing the number of states is the most straightforward way to make a model more desirable from a solution point of view; this is the approach taken in the work on weak isomorphism in PEPA.

Note that in general, when model manipulation is based on an equivalence relation which captures all relevant aspects of the behaviour of a model the subsequent solution of the transformed model will be exact [33,29,4]. However, when a partial order relation or an equivalence relation which does not consider all aspects of represented behaviour is used, model manipulation may result in an approximation of the original model. This is the case when weak bisimulation is used with respect to a subset of TIPP in Mertsiotakis's work on throughput approximation (see Section 6.2): the weak bisimulation relation cannot capture the timing characteristics of the models.

In the context of model manipulation, the role of compositionality is perhaps secondary to the equivalence relations which are used to define the model transformations. Nevertheless it is an important role, distinguishing the use of the model manipulation techniques in the MPA setting from their direct application at the Markov process level. Using process algebra apparatus we can establish the congruence of an equivalence relation as a feature of a language. The consequences of this form of model manipulation are significant: components within the model may be manipulated, and *improved*, in isolation. The modified model is formed as the composite of the modified components. Thus the state space of the complete model may never need to be constructed [33,32,29]. This greatly reduces the complexity of the procedure and ultimately, may make intractable models tractable.

Another approach which has been taken to exploit MPA model compositionality during Markov process generation, is the use of tensor algebra. Again, the

objective is to tackle the problem of state space explosion but the strategy is to alter the representation of the underlying Markov process. Instead of capturing the Markov process as a single infinitesimal generator matrix, using tensor algebra it is represented as an expression of smaller matrices. Specially designed algorithms are able to take advantage of this expression and find the steady state solution in terms of the expression, avoiding the construction of the complete matrix. This general approach has been pioneered by Plateau and others with *Stochastic Automata Networks* [53,54,58].

In [12], Buchholz identifies the relationship between the parallel composition operator in the SPA language, *MPA*, and tensor algebra expressions for the underlying Markov process. He shows how an expression for the complete model can be constructed in terms of smaller matrices representing the individual components and the synchronisation sets in operation between them. However, the form of synchronisation defined in the semantics of *MPA* is slightly unusual and tailored to the tensor representation. Similarly, in [55], Rettelbach and Siegle construct a minimal compositional semantics for a subset of TIPP, called TIPP^{MS}. Specifically, this language includes a synchronising replication operator but not parallel composition in its general form. In this work a matrix is defined for each language expression without recourse to an operational semantics and the associated labelled transition system. This is achieved by constructing the matrix from sub-matrices corresponding to terms in the expression using matrix operators corresponding to the process algebra combinators. In particular, the replication operator maps to the tensor sum of replicated copies of the matrix corresponding to the replicated process.

More recently, in [36], Hillston and Kloul present a representation of the Markov process underlying a PEPA model in terms of a tensor product of terms. Unlike the earlier work by Buchholz, Rettelbach and Siegle, this is not based on an especially designed, syntactically restricted, form of the process algebra. Whilst the representation is similar to previous representations of Stochastic Automata Networks and Stochastic Petri Nets, it has novel features, arising from the definition of the PEPA models. In particular, capturing the correct timing behaviour of cooperating PEPA activities relies on a new tensor operator defined within the paper [36].

We do not classify this work as decomposed solution since the underlying Markov process is still solved as a single entity although it is represented in a decomposed form. In the following sections we survey work which we classify as truly decomposed solution. In these cases the MPA model is used to generate not one Markov process but several, and these processes are solved separately.

4 Decomposed Solution

A variety of decompositional or structural techniques have been proposed to aid in the solution of large Markov processes. Recently several results have been published which show that, at least for some particular cases, there is a clear relationship between these techniques and MPA model descriptions. In the fol-

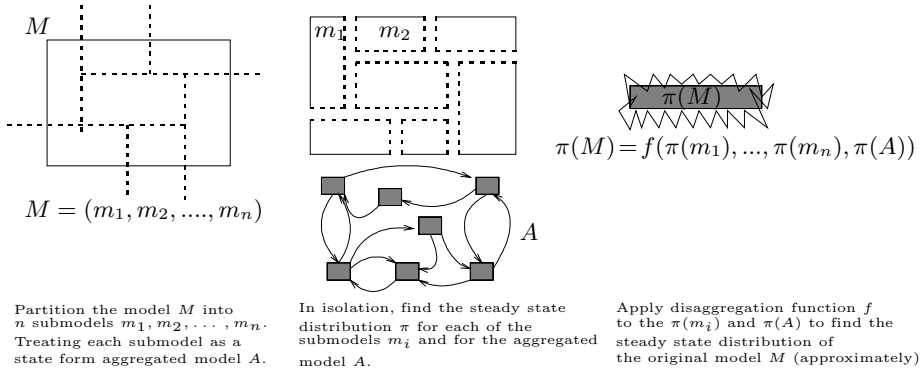


Fig. 2. Schematic overview of decomposed solution

lowing two sections we survey some of these recent results. In this section we aim to provide a general introduction and framework for decomposed solution of MPA models.

The overall approach is summarised schematically in Figure 2. A model is considered to be made up of a number of submodels $m_1, m_2, \ldots, m_n$, and any state of the model can be expressed in terms of the local states in each of the submodels. Note that in general these submodels will correspond to regions of the state space of a model and may or may not correspond to components within the system being represented. Although they may not be independent, a decomposed solution will typically solve each of these submodels in isolation, generating a corresponding steady state distribution $\pi_i(\cdot)$ over the state space of the submodel m_i. In addition an *aggregated* model may also be formed, aiming to capture the pattern of interactions between submodels. In this model each submodel is represented by a single state and the transitions between states are intended to capture the points when the complete model moves between submodels (regions of the state space). A steady state distribution for this aggregated model is also generated. An approximate steady state distribution of the original model is then obtained by applying a *disaggregation function* to the results of the separate analyses. How the submodels are identified, the formation of the aggregated model and the definition of the disaggregation function are all particular to the decomposed solution technique.

Many such techniques are well-known at the Markov process level. The advantage of characterising the corresponding class of MPA models is that by "lifting" the definition from the stochastic process level to a formally defined high-level modelling paradigm we can facilitate the automatic detection of these structures when they occur, thus avoiding the construction of the original Markov process.

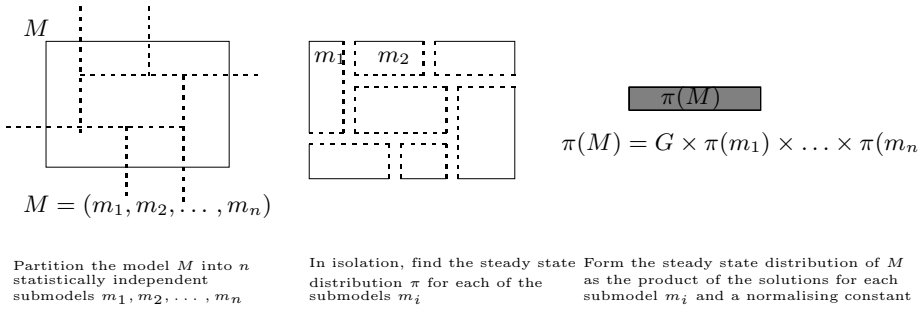


Fig. 3. Schematic overview of product form solution

4.1 Product Form Solutions

It is clear that there is great advantage to be gained if the compositional structure of a MPA model can be used during model solution. This would mean that the submodels within the general scenario shown in Figure 2 would correspond directly to components within the MPA model, i.e. the components of the MPA model could be solved separately and their solutions combined to obtain a solution, exact or approximate, of the whole Markov process. One class of Markov processes which are susceptible to such an efficient solution technique are those which exhibit a *product form* steady state distribution. Significant effort has gone into identifying those classes of PEPA models in which the constituent components will generate a product form solution.

Consider a Markov process $X(t)$, whose state space $\mathcal{S}$ is of the form $\mathcal{S} = S_1 \times S_2$, i.e. each state $\boldsymbol{s} = (s_1, s_2)$ contains two pieces of information capturing different aspects of the current state. In general, these aspects may be related in many ways. When the process $X(t)$ exhibits a product form solution, i.e. $\pi(\boldsymbol{s}) = \pi_1(s_1) \times \pi_2(s_2)$, it indicates that these different aspects of the state description are independent. In this case the general scenario simplifies to that shown in Figure 3. Note that the aggregate model is no longer constructed or solved, and that disaggregation function is a simple multiple of the steady state probabilities of the submodels. Moreover this technique generates an *exact* solution up to a normalisation constant.

Product form distributions have been widely used in the analysis of queueing networks and, due to their efficient solution, have contributed to the popularity of queueing networks for performance analysis. For example, Jackson networks [42] and their generalisation, BCMP networks [3], have been widely employed. Here the underlying Markov process is known to have a reversible or quasi-reversible structure.

In contrast stochastic Petri nets (SPN) have rarely been found to be amenable to such efficient steady state solution, except when some of the expressiveness of the formalism is reduced, for example by excluding resource sharing and competition over resources in a general form. By imposing these restrictions, Henderson

and Taylor develop product form over the *places* of the Petri net, to obtain a product form similar to that obtained for queueing networks [27]. Lazar and Robertazzi establish a first step towards a product form over *subnets*, characterising independence between subnets which compete for resources [45]. Donatelli and Sereno show how both these approaches are related to T-semiflows in the Petri net [20]. An excellent survey of product form results for Petri nets can be found in [57].

Work on finding PEPA models which give rise to product form solutions has drawn on the previous work on both queueing networks and SPNs. Essentially this can be seen as an investigation of when components *interact* and yet remain *statistically independent*. It is clear that when any MPA model consists of completely independent sequential components, i.e. $P \parallel Q$, the equilibrium distribution will have a product form:

$$\pi(P \parallel Q) = \pi_P(P) \times \pi_Q(Q) \tag{1}$$

where π_P and π_Q are the steady state distributions over the local states of P and Q respectively. However few real systems consist of components which are independent in this way, and if they did the state space explosion problem would not arise because it would be obvious that the components could be analysed separately. The challenge is to find circumstances in which components P and Q which synchronise, $P \underset{L}{\bowtie} Q$ in PEPA notation, still exhibit statistical independence.

4.2 Other Decomposed Solution Techniques

In the following section we consider some techniques for *aggregated decomposed* solutions. Here, it is not simply a case of splitting the model into submodels or components in the style of product form. As well as a stochastic representation of each of the components, the decomposed solution involves a stochastic representation of the interactions between these components, the *aggregated model*. In most cases these stochastic representations will be Markov processes but in the work by Bohnenkamp and Haverkort on decomposition via synchronisation points semi-Markov processes are used [8]. In addition to work on characterising the class of MPA models for which the technique is appropriate, the research effort can also involve development of a good disaggregation function, which allows the results of the individual submodels and the aggregated model, to combined to approximate the solution of the original model. In several cases this effort has been based on previous work on SPN.

5 Product Form PEPA Models

5.1 Reversibility

Informally, a reversible Markov process is one which behaves identically when we observe it with time reversed as when we observe it with time flowing forward. At

the Markov process level there are several ways to characterise these processes, but we state only the *local balance* condition. An irreducible, stationary Markov process $X(t)$ is *reversible* if it satisfies the detailed balance equations:

$$\pi(j)q(j,k) = \pi(k)q(k,j) \tag{2}$$

where $q(j,k)$ is the instantaneous transition rate from state j to state k and $\pi(\cdot)$ is the steady state probability distribution. A more complete account of reversibility can found in the book by Kelly [44].

An initial study of MPA models giving rise to reversible Markov processes was presented by Bhabuta *et al.* in [5]. This paper largely considered the problem at the level of the underlying state space. In [38], Hillston and Thomas, identify syntactic conditions which a PEPA model must satisfy in order for the underlying process to be reversible. The problem is tackled in two stages. First, a basic class of sequential components which give rise to reversible structures are identified. These are shown in Figure 4. Then, assuming that a known class of reversible PEPA components exist, the authors investigate under what circumstances the conditions for reversibility will be preserved if reversible components are composed using the combinators of PEPA.

Fundamental to the basic class of reversible sequential components is the notion of a *reverse pair*. A pair of action types $(\alpha, -\alpha)$ form a reverse pair if, in any state, any α transition leads to a state in which a $-\alpha$ transition leading back to the original state. This ability to "undo" any transition in the subsequent transition seems to be fundamental to reversibility. It clear to see that this is a necessary condition for equation (2) to be satisfied. From this starting point various canonical forms for sequential reversible components are described. The interesting conditions for when reversible components can be composed without losing the reversibility property relate to cooperation. In [38] detailed conditions, on the form of the cooperation set and the rates of the activities involved, are given. We refer the reader to that paper for more detail.

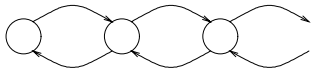
5.2 Quasi-Reversibility

Like reversibility, the origins of *quasi-reversibility* are in queueing theory. It is the condition which allows a wide class of queueing networks to be separated into their individual queues and solved in isolation, provided traffic equations are solved to give appropriate arrival rates at each queue. Formally, a stationary Markov process $X(t)$ is *quasi-reversible* if, for all times t_0 the state $X(t_0)$ is independent of

1. the input process after t_0 and
2. the output process before t_0.

Rather than the detailed balance equations which characterised reversibility, a quasi-reversible process satisfies *partial balance equations*:

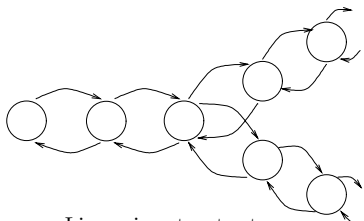
$$\pi(i) \sum_{j \in S'} q(i,j) = \sum_{j \in S'} \pi(j)q(j,i) \tag{3}$$



Birth-death process

$$Gambler_0 \stackrel{\text{def}}{=} (borrow, r).Gambler_1$$
$$Gambler_n \stackrel{\text{def}}{=} (win, w).Gambler_{n+1} + (lose, l).Gambler_{n-1} \quad n \geq 1$$

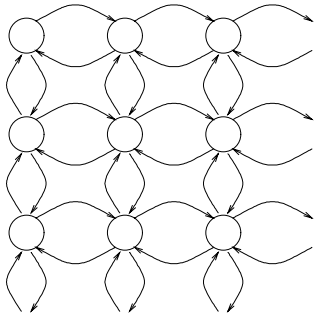
Birth-death component



Linear input-output process

$$Gambler_0 \stackrel{\text{def}}{=} (borrow, r).Gambler_1$$
$$Gambler_1 \stackrel{\text{def}}{=} (win, w).Gambler_2 + (lose, l).Gambler_0$$
$$Gambler_2 \stackrel{\text{def}}{=} (win_1, w_1).Cards_3 + (win_2, w_2).Slots_3 + (lose, l).Gambler_1$$
$$Cards_3 \stackrel{\text{def}}{=} (win, w).Cards_4 + (lose, l).Gambler_2$$
$$Slots_3 \stackrel{\text{def}}{=} (win, w).Slots_4 + (lose, l).Gambler_2$$
$$Cards_n \stackrel{\text{def}}{=} (win, w).Cards_{n+1} + (lose, l).Cards_{n-1}$$
$$Slots_n \stackrel{\text{def}}{=} (win, w).Slots_{n+1} + (lose, l).Slots_{n-1} \quad n \geq 4$$

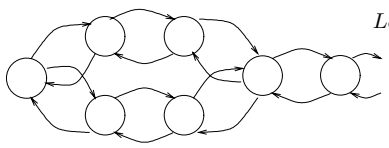
Linear input-output component



Parallel input-output process

$$Gambler1_0 \stackrel{\text{def}}{=} (borrow, r).Gambler1_1$$
$$Gambler1_n \stackrel{\text{def}}{=} (win, w).Gambler1_{n+1} + (lose, l).Gambler1_{n-1} \quad n \geq 1$$
$$Gambler2_0 \stackrel{\text{def}}{=} (borrow, r).Gambler2_1$$
$$Gambler2_m \stackrel{\text{def}}{=} (win, w).Gambler2_{m+1} + (lose, l).Gambler2_{m-1} \quad m \geq 1$$
$$Gamblers_{(n,m)} \stackrel{\text{def}}{=} Gambler1_n \parallel Gambler2_m$$

Parallel input-output component



Symmetrical branch-join process

$$Loser_0 \stackrel{\text{def}}{=} (borrow, r_1).BJ_1 + (borrow, r_2).21_1$$
$$BJ_1 \stackrel{\text{def}}{=} (win, w).BJ_2 + (lose, l).Loser_0$$
$$BJ_2 \stackrel{\text{def}}{=} (win, w).Spin_3 + (lose, l).BJ_1$$
$$21_1 \stackrel{\text{def}}{=} (win, w).21_2 + (lose, l).Gambler_0$$
$$21_2 \stackrel{\text{def}}{=} (win, w).Spin_3 + (lose, l).21_2$$
$$Spin_3 \stackrel{\text{def}}{=} (win, w).Spin_4 + (lose, l_1).BJ_2 + (lose, l_2).21_2$$
$$Spin_n \stackrel{\text{def}}{=} (win, w).Spin_{n+1} + (lose, l).Spin_{n-1} \quad n \geq 4$$

Symmetrical branch-join component

Fig. 4. Basic reversible structures

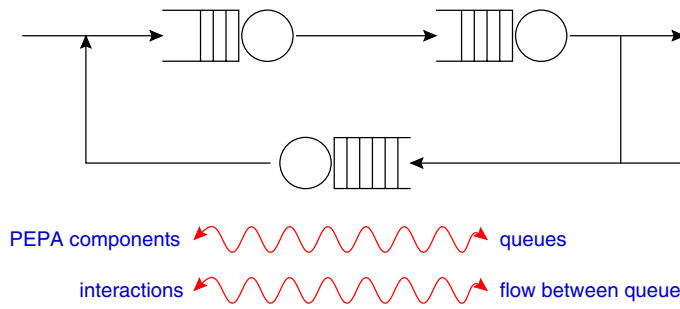


Fig. 5. Relating queueing network concepts to PEPA

for all states i and a corresponding subset of states S'. Again, more details of the definition of quasi-reversibility can be found in the book by Kelly [44].

In [25], a PEPA characterisation of this class is presented. As in the work on reversibility, the approach is to first find simple instances of PEPA processes which give rise to quasi-reversible structure in their underlying Markov process (*QR components*). Then, conditions are established under which these components can be composed whilst maintaining the quasi-reversible property. Relative to the simple product form PEPA case presented in equation (1), this does allow interaction between the components P and Q. But strong restrictions are placed on the form of this interaction. Again the notion of a *reverse pair* is important. Intuitively, it is easy to see the reverse pair action types as a generalisation of the input-output behaviour of a queue.

Not surprisingly, given the origins in queueing networks, the form of admissible PEPA interaction is a *flow cooperation*. This means that the "positive" half of a reverse pair in one component is carried out in synchronisation or cooperation with the "negative" half of a reverse pair in another. The "positive" actions correspond to the input process in the definition of quasi-reversibility, while the "negative" correspond to the output process. The subsequent theorems, stated below and developed in more detail in [25], correspond to those for open and closed queueing networks reported in [44].

Theorem 2 (Open Quasi-reversible Interactions). *An open flow cooperation of QR components has the following properties at steady state:*

- *The marginal states of the individual components are independent.*
- *For an individual component the steady state distribution and the distribution over states at the time of completion of a positive action of a given type are identical and are both as they would be if the component were in isolation with independent Poisson external sources with rates given by traffic equations.*
- *Under time reversal the model becomes another flow cooperation of QR components; in particular sources become sinks and vice versa.*
- *The underlying Markov process is quasi-reversible. Thus the completion of negative actions of each type at the sinks form independent Poisson processes*

and the state of the model at time t_0 is independent of the completion instants of the negative actions in the model prior to time t_0.

Theorem 3 (Closed Quasi-reversible Interactions). *A closed flow cooperation of QR components, $C_1, C_2, \ldots, C_m$ has the following properties at steady state:*

- *Let $\pi(C_{k_i})$ be the steady state probability that component C_k is in state C_{k_i}, $1 \leq k \leq m$, when interacting only with its Poisson external sources with rates given by the traffic equations. Then the steady state probability distribution of the closed flow cooperation is*

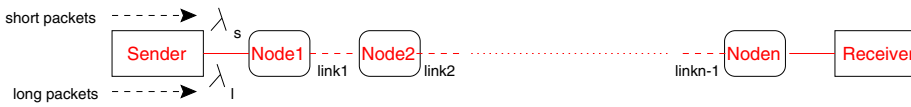
$$\prod_{i=1}^{m} \pi(C_{k_i})/B$$

  where B is the normalising constant.
- *The distribution over states at the time of completion of an positive action of type α, which produces an $(\alpha, r)^n$-derivative in a component of the state vector representation, is identical to that above for the closed flow cooperation obtained by reducing n to $n-1$.*
- *Under time reversal the model becomes another closed flow cooperation of QR components.*
- *The underlying Markov process is quasi-reversible. Thus the completion of negative actions of each type at the sinks form independent Poisson processes and the state of the model at time t_0 is independent of the completion instants of the negative actions in the model prior to time t_0.*

Figure 6 shows a simple packet transmission network expressed as a PEPA model. This model satisfies the quasi-reversibility characterisation. Each node is a QR component, enabling two positive actions on its outgoing links and two corresponding reverse actions on its incoming links. The cooperation between $Node k$ and $Node(k+1)$ is a flow cooperation since a positive action in one cooperates with a negative action in the other. The sender is a source for the system and the receiver is a sink. Thus we have an open quasi-reversible interaction. As a consequence of the characterisation we know that the steady state distribution of the complete model can be expressed as a product of terms, one corresponding to each node in the network, as shown.

5.3 Routing Process Approach

Sereno's work, reported in [56], derives product form criteria for PEPA models based on earlier work on product form criteria for SPN [26,27,11]. The SPN results rely on defining a Markov chain whose states correspond to the transitions of the SPN, the so-called *routing chain*. The condition for this chain to exist is that the set of places into which tokens are placed when a transition fires should be exactly the input places of another transition. This condition places severe



$$System \stackrel{\text{def}}{=} Sender \underset{\{snd_{sht}, snd_{lg}\}}{\bowtie} Network \underset{\{dlvr_{sht}, dlvr_{lg}\}}{\bowtie} Receiver$$

$$Network \stackrel{\text{def}}{=} Node1_{0,0} \underset{\{lk^1_{sht}, lk^1_{lg}\}}{\bowtie} Node2_{0,0} \underset{\{lk^2_{sht}, lk^2_{lg}\}}{\bowtie} \cdots \underset{\{lk^{n-1}_{sht}, lk^{n-1}_{lg}\}}{\bowtie} Noden_{0,0}$$

$$\vdots \quad \vdots \qquad \vdots$$

$$Node_{i,j} \stackrel{\text{def}}{=} (lk^{k-1}_{sht}, \top).Node_{i+1,j} + (lk^{k-1}_{lg}, \top).Node_{i,j+1} + (lk^{k}_{sht}, \mu_{k1}).Node_{i-1,j} + (lk^{k}_{lg}, \mu_{k2}).Node_{i,j-1}$$

$$\vdots \quad \vdots \qquad \vdots$$

$$\pi\left(Node1_{i_1,j_1}, \ldots, Node_{i_k,j_k}, \ldots, Noden_{i_n,j_n}\right) = \prod_{k=1}^{N} \left[\left(\frac{\lambda_s}{\mu_{k1}}\right)^{i_k} \left(\frac{\lambda_l}{\mu_{k2}}\right)^{j_k} \left(1 - \frac{\lambda_s}{\mu_{k1}}\right) \left(1 - \frac{\lambda_l}{\mu_{k2}}\right) \right]$$

Fig. 6. Quasi-reversibility example: A simple packet transmission network

restrictions on the forms of synchronisation and resource contention which can be represented in the net.

In [56], Sereno uses a vector representation of the state of a PEPA model[2] in the characterisation of the class of models which have a product form based on the *routing process*. It is assumed that some preprocessing of the model is done in order to collect information and to aggregate the model, using one of the techniques outlined in Section 3. The information which is needed is the local state space of each component, and, for each action type of the model, which local states of participating components enable the action and which appear after it has been performed. These two sets of local states are called the *pre-set* and *post-set* of the action, respectively. The state vector representation is composed of sub-vectors, one corresponding to each defined sequential component; an element within the sub-vector corresponds to a local state within corresponding component. In the representation of any particular state the value of an element within the vector records the number of instances of each local state exhibited in the current syntactical state of the model. Storing the state in this form, together with the pre- and post-sets of actions represented as vectors, allows the effect of completing an action to be written down in vector form.

[2] Note, however, that this differs from the state vector defined in Definition 5.

There are several restrictions placed on the PEPA models, in particular with respect to action types. An action can only have one pre-set—this implies that each action within the behaviour of a sequential component must have a distinct name. Moreover if actions of the same type occur within different sequential components they must be synchronised.

Sereno's approach for PEPA is completely analogous to the earlier work on SPN—he defines a Markov chain in which the states correspond to the actions of the PEPA model. This is called the *routing process*. The global balance equations of the routing process correspond to the traffic equations of queueing networks. If the state space of this process can be partitioned into equivalence classes of enabling actions (roughly speaking, one action *enables* another if the post-set of one is the pre-set of the other; we take the transitive closure of that relation), then a product form solution exists. Moreover the partition forms the basis for the decomposition.

5.4 Product Form over Submodels

As mentioned earlier, in [45] Lazar and Robertazzi investigate a product form in the context of SPN—the decomposition is carried out over subnets, which may still need to be solved by standard numerical techniques to find their local steady state.

In [10], Boucherie generalised their result and characterised the class of underlying Markov processes. Such a process is formed as the product of a set of Markov processes which compete over a set of resources. The resources are not explicitly represented but the competition has two important impacts on the state space and the transition rates of the product process. Firstly, if two constituent processes compete over a resource they cannot both enter the region of their state space representing possession of the resource at the same time. Thus areas of the state space of the product process are eliminated. It is assumed that the product process will change state in only one of the constituent processes at each state change. The second effect of the competition over resources is to limit this still further—if two processes compete over a resource, and one of them is currently holding the resource, then the other cannot make any state change, no matter where it is in its state space. Thus when a constituent process holds a resource, all competing processes are blocked.

This form of interaction is illustrated by the product process shown in Figure 7. Here we assume that we have two Markov processes S_1 and S_2. The state space of each process is partitioned into those states in which no resource is being used (A_{i0}) and those in which the resource r is being used (A_{ir}). Immediately we see that the bottom right hand corner of the product state space has been eliminated because the processes cannot access the resource r simultaneously. Moreover, when, say, S_1 holds the resource (is in partition A_{1r}), we see that process S_2 is unable to make any transitions at all, not even those not involving resource use.

Essentially the product form result holds because in each state of the product process each constituent process satisfies its own global balance equations. If it

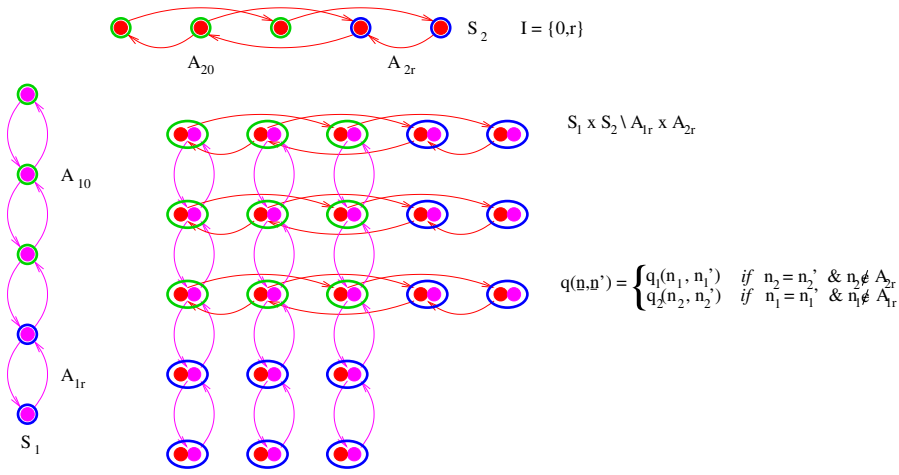


Fig. 7. Simple example of two-dimensional resource contention Markov process

can make a transition it is free to act as if it were independent; alternatively, it is completely blocked and satisfies its global balance equations trivially.

In [34,39], Hillston and Thomas characterise this class of Markov processes in PEPA. The class of models that they identify consist of independent components, which give rise to the constituent Markov processes of the underlying Markov process. These components interact indirectly by synchronisation with *resource components*. Compared with the simple product form model presented in equation (1), the general form of these process algebra terms and the resulting product form is, schematically:

$$\pi\left((P \parallel Q) \underset{L}{\bowtie} R\right) = B \times \pi_P(P \underset{L}{\bowtie} R) \times \pi_Q(Q \underset{L}{\bowtie} R) \tag{4}$$

where the component R represents the resource, π_P and π_Q are the steady state distributions over the derivatives of $P \underset{L}{\bowtie} R$ and $Q \underset{L}{\bowtie} R$ respectively, and B is the normalising constant. The decomposition is formed by considering each of the model terms (P and Q in this case) acting in synchronisation with the resource (R) in isolation.

In PEPA, a component is termed a *resource* if it is never free to act independently; all its activities must be carried out in synchronisation with the rest of the model. All components are assumed to have cyclic behaviour. In this context a component is considered to be using or holding the resource if it has carried out the first action of the resource's behaviour in synchronisation with the resource. The semantics of PEPA ensure that the state space of the product process is suitably modified, i.e. that two competing components cannot hold the resource simultaneously. In order to ensure that the correct condition is also satisfied by the transition rates of the product process, Hillston and Thomas place

a further restriction on the cooperation set in operation between the resource component and the rest of the model. If a model component wishes to use the resource during one of its possible behavioural cycles, it must gain control of the resource at the start of the cycle and release it only at the end. This guarantees that other competing components will be blocked when the component holds the resource. Although presented here informally, these conditions are defined as formal syntactical conditions which can be checked on the model specification [39].

Any resource component is redundant within the state vector representation of a model, in the sense given in Definition 6. This means that the resource can be eliminated from the state vector and the process will then satisfy the condition that only one constituent component changes its local state for each global transition.

The product form PEPA models discussed in previous subsections have been based on components of a particular structure which interact in a restricted way, preserving a form of independence between the components. Here the characterised models represent the competition of otherwise independent components over resources. The form of these components is not restricted; however they do place a relative restriction on the form of the resource and on the form of the cooperation set in operation between the resource and the rest of the model. As suggested by equation (4) above, these components, together with the resource, are solved in isolation, these partial solutions subsequently being combined to give a solution of the complete model. The outstanding problem of this approach, however, is the calculation of the normalising constant.

Figure 8 shows a PEPA model which satisfies the criteria for this form of product form. Consider a simple railway system with the arrangement of tracks and stations as shown below. There are two trains: $Train_1$ which circulates round stations A, B and C, and $Train_2$ which has a choice of two routes, either round D, B, and C, or round D, E and F (see Figure 9).

The trains are independent of each other but must compete for access to the shared piece of track, as controlled by the signal. In this model the signal is the resource component. It is straightforward to see that it satisfies the condition that it never acts independently, and that its local state can always be deduced from the local states of the two trains. Moreover, for each train, any cycle of behaviour involving the signal can only begin when cooperation with the signal is possible, and ends by *releasing* the signal again.

The models presented in [10], including those presented as SPN, relied on the insight of the modeller to detect the product form structure. Moreover, in the case of the SPN models, a non-standard state representation had to be used in order to eliminate the resource from the model representation. The PEPA models do not have this disadvantage since the resource may (indeed, must) be represented explicitly and subsequently eliminated from the state representation using formally defined procedures.

$$
\begin{aligned}
Train_{1A} &\stackrel{\text{def}}{=} (track_{AB}, t_1).(stop_B, s_1).Train_{1B}\\
Train_{1B} &\stackrel{\text{def}}{=} (track_{BC}, t_1).(stop_C, s_1).Train_{1C}\\
Train_{1C} &\stackrel{\text{def}}{=} (track_{CA}, t_1).(stop_A, s_1).Train_{1A}\\
Train_{2D} &\stackrel{\text{def}}{=} (track_{DB}, t_2).(stop_B, s_2).Train_{2B} + (track_{DE}, t_2).(stop_E, s_2).Train_{2E}\\
Train_{2B} &\stackrel{\text{def}}{=} (track_{BC}, t_2).(stop_C, s_2).Train_{2C}\\
Train_{2C} &\stackrel{\text{def}}{=} (track_{CD}, t_2).(stop_D, s_2).Train_{2D}\\
Train_{2E} &\stackrel{\text{def}}{=} (track_{EF}, t_2).(stop_F, s_2).Train_{2F}\\
Train_{2F} &\stackrel{\text{def}}{=} (track_{FD}, t_2).(stop_D, s_2).Train_{2D}
\end{aligned}
$$

$$
\begin{aligned}
Signal &\stackrel{\text{def}}{=} (track_{AB}, \top).Signal_1 + (track_{DB}, \top).Signal_2\\
Signal_1 &\stackrel{\text{def}}{=} (track_{CA}, \top).Signal\\
Signal_2 &\stackrel{\text{def}}{=} (track_{CD}, \top).Signal
\end{aligned}
$$

$$
\begin{aligned}
Railway &\stackrel{\text{def}}{=} \left(Train_{1A} \parallel Train_{2D}\right) \underset{L}{\bowtie} Signal\\
\text{where } L &= \{track_{AB}, track_{CA}, track_{DB}, track_{CD}\}
\end{aligned}
$$

Fig. 8. A small railway system

5.5 Queueing Discipline Models

Queueing discipline models were introduced by Clark in [15] and presented in more detail in his recent thesis [16]. The initial motivation for Clark's work was the investigation of when a PEPA model could be shown to be *insensitive*[3] to the distribution governing activity duration. In this way he intended to increase the expressiveness of PEPA by removing the restriction to exponential distributions for some activities within appropriately structured models. It is well-known from queueing theory that there is a close link between insensitivity and product forms, so it is perhaps unsurprising that Clark's insensitive structures also give rise to models which have a product form steady state solution.

Queueing discipline models are constructed using a new PEPA combinator $Q_{A,\xi}(\cdot)$, the *queue discipline combinator*. This is a *derived* combinator meaning that it can be defined in terms of the existing combinators of the language and so does not necessitate any modification of the semantics of the language. In the combinator syntax the set A denotes a set of *queueing actions* and ξ denotes a set of rates; the argument to the combinator is a set of independent, complete

[3] When a Markov process is *insensitive* its steady state distribution is unaffected by the form of distribution used to determine state sojourn times, and affected only by the mean of such distributions.

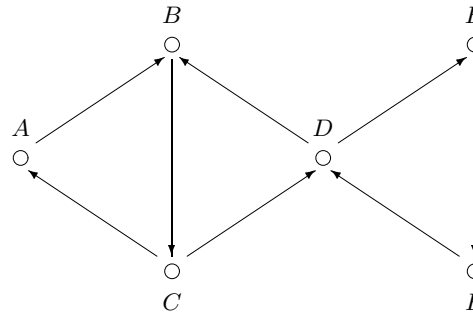


Fig. 9. Schematic representation of the small railway system

sequential PEPA components, $S_1, \ldots, S_n$. For example, in a model in which the set of rates is left unspecified, the derived form is:

$$Q_A(S_1, \ldots, S_n) \stackrel{\text{def}}{=} (S_1 \parallel \cdots \parallel S_n) \underset{M_A}{\bowtie} R_A$$

for suitably chosen M_A (the *arbiter synchronisation set*) and R_A (the *arbiter*).

Intuitively, the components S_i proceed in parallel except when they wish to carry out an activity whose type appears in A. When this occurs the component must *queue* to carry out the queue action, possibly waiting for other components queueing on the same action to complete it first. The arbiter imposes the necessary FCFS queueing discipline. This is shown schematically in Figure 10.

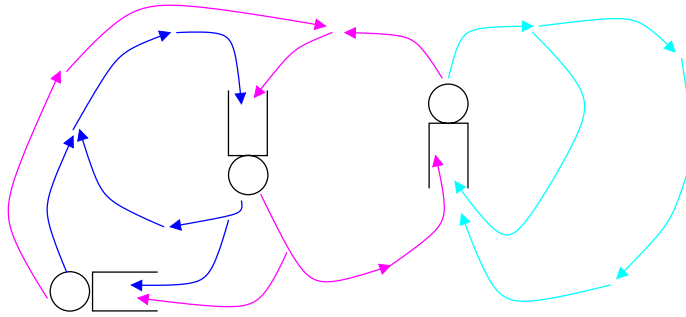


Fig. 10. Schematic view of queueing discipline models

If a model is constructed in this way, Clark has shown that it will have a product form steady state solution, as stated in the following theorem (see [16] for more details).

Theorem 4 (Solution of queueing discipline PEPA models). *Consider* $P \equiv Q_\chi(S_1, \ldots, S_n)$*, assuming that there are* N *queues and that the rate of*

leaving queue j when there are i processes waiting is ξ_{ij}. Then the steady state solution of P is given by

$$\pi(P) = B \prod_{i=1}^{n} \pi_i(S_i) \cdot \prod_{i=1}^{N} \prod_{j=q_i+1}^{d_j} \xi_{ij} \cdot \prod_{\substack{i=1 \\ S_i \in \Theta}}^{n} \sum_{(\alpha,r) \in \mathcal{A}ct(S_i)} r$$

where B is a normalising constant and Θ is the set of processes currently queueing.

One of the major differences between this class of product form models and those considered earlier in the section is that the product form of the solution of the models is guaranteed by construction, and does not have to checked against a syntactic characterisation. In other words the queueing discipline combinator can be regarded as a product form combinator, i.e. it specifies interaction between components in such a form that their independence, in terms of the steady state distribution, is preserved.

Figure 11 shows a PEPA model of a database locking system, such as might be found in a transaction processing system. This model is constructed using the new combinator. It consists of components to represent a collection of transaction classes. A manager, which enforces locking rules on database objects is represented implicitly by the queueing discipline and the queueing activities. A transaction in the model may either attempt to

- modify a particular database object, in which case it attempts to acquire a *write* lock for that object; or
- read a particular object in its current state, in which case it does so regardless of any locks present on that object; this is called a *dirty read*.

This behaviour can be seen in the representation of the transactions.

The model has a product form steady state distribution as described by Theorem 4. Moreover, Clark's insensitivity result means that the *think* activity associated with each transaction class is not restricted to have an exponentially distribution. For example, the duration of this activity can be set to be deterministic.

The product form solution of the queueing discipline structure suggests a link to the well-known class of BCMP queueing networks [3]. These PEPA models appear to capture infinite server and FCFS service stations. However, note that the PEPA model views the model *from the customer's perspective*, i.e. the behaviour of the system is captured as a set of states recording the point in its cycle round the system that each customer has reached. Using this view of the system, intuitively speaking, each non-queueing activity corresponds to an infinite server station, whilst each queueing activity corresponds to a FCFS service station. More explanation of this mapping can be found in [17].

$$TxnC_i \stackrel{\text{def}}{=} (think_i, t_i).Txn_i$$

$$Txn_i \stackrel{\text{def}}{=} \sum_{j=1}^{n} (work, ww \times p_j).(writelock_{ij}, r_{ij}).(commit_{ij}, wl_{ij}).Loop_i$$
$$+ (work, wr).(dirtyread_i, r).(accessdb, rl).Loop_i$$

$$Loop_i \stackrel{\text{def}}{=} (nexttxn_i, n \times a_{ij}).TxnC_i + (nexttxn_i, (1-n) \times a_{ij}).Txn_i$$

$$A_j = \{commit_{ij}, j \leq i \leq M\} \text{ for } 1 \leq j \leq n$$

$$TPS \stackrel{\text{def}}{=} Q_{A_1}(\ldots Q_{A_n}(TxnC_1, \ldots, TxnC_M)\ldots) \equiv Q_\xi(TxnC_1, \ldots, TxnC_M)$$

Fig. 11. Transaction processing system model with queueing discipline structure

5.6 Quasi-Separability

A quasi-separable Markov process does not have a product form solution in the sense of the other classes of models considered in this section. However we discuss this approach here because it has more similarities with the product form cases than with the aggregated decomposed solutions considered in the following section. Unlike the case with product form processes it is not possible to find the exact solution of the steady state distribution of a quasi-separable process as a product of the local steady state distributions. Nevertheless decomposed solution can lead to exact results for the local steady state distributions and many performance measures, and no aggregated model needs to be formed.

As with reversibility, quasi-reversibility and queueing discipline, the notion of quasi-separability is one which has been developed in relation to queueing networks, in particular queueing networks in which breakdowns occur [51,50]. It is assumed that the Markov process is comprised of a number of components and that there are two pertinent pieces of information for each component. A representation of the whole process can then be formed as a pair of vectors, each vector capturing one piece of information for each component. For a process to be *quasi-separable* it must be possible to analyse the behaviour of a component, say component i, given the ith element of the first vector and all elements of the second, or vice versa. This allows the complete process to be reduced to a number of sub-models, each of which contains all the information about exactly one component. In the queueing network context the two pieces of information about each queue are typically its operational state and the number of customers present.

In [60], Thomas and Gilmore present a characterisation of PEPA models which are quasi-separable. It is assumed in this characterisation that the infor-

mation which must be included in each decomposed submodel is not distributed between the components but maintained by a single *scheduler* component. There are several conditions on the way in which this component may interact with the other components of the model, which do correspond to the components of the system. When the scheduler changes its state it must do so by an individual action, or by a shared action in which the other participant is passive. Furthermore the individual components have no direct interaction between them—they must be in parallel composition with no synchronisation. In other words, each of these components interacts only with the scheduler. This bears some similarities both with the scenario for product form over submodels and for the queueing discipline structure. However, note that the behaviour of the scheduler and the resource, in relation to the rest of the model, are quite different. The model is decomposed into a set of models, each comprising of a single component considered with the scheduler, in isolation. More details can be found in [60,59].

6 Aggregated Decomposed Solutions of MPA Models

6.1 Time Scale Decomposition

The work on time scale decomposition in MPA is based on the notion of *near completely decomposable* Markov processes [18], and inspired by previous work on time scale decomposition of SPN models [6,2]. A characterisation of a near completely decomposable Markov process at the matrix level is that the matrix is block structured with elements in the diagonal blocks being at least an order of magnitude larger than elements in the off-diagonal blocks. This implies that the model is made up of subsystems whose internal interactions are much more frequent than the interactions between subsystems. As a consequence it can be assumed that the subsystems reach an internal equilibrium between external interactions. Thus a steady state for each Markov process corresponding to a diagonal block of the original process is found; the interactions are modelled by an aggregated model capturing the interactions between subsystems as represented by the off-diagonal blocks. The aggregated model has one state for each subsystem/diagonal block. There are known error bounds for the technique based on the magnitude of the largest element in an off-diagonal block.

The initial classification of MPA models susceptible to time scale decomposition [37], relied on a classification of the sequential components within a model into *fast* or *slow*; this in turn was based on a classification of all actions relative to some threshold rate. A component is considered to be fast if it enables fast or passive actions; a component is considered to be slow if it enables only slow actions. Only models consisting solely of fast and slow components could be analysed. The state of such a process was represented by its state vector, ordered in such a way as to emphasise the distinction between fast and slow components. Thus a model P which is comprised of k fast and ℓ slow components may be represented as: $P \equiv (F_1, \ldots, F_k, S_1, \ldots, S_\ell)$. Then each decomposed component corresponds to a set of state vectors which exhibit the same derivatives in all the slow components:

$$A_{[S_1,\ldots,S_\ell]} \equiv \{(F'_1,\ldots,F'_k,S'_1,\ldots,S'_\ell) \mid S'_1 \equiv S_1,\ldots,S'_\ell \equiv S_\ell\}$$

The elements of this set are found using the semantics of the language when the original model is considered in composition over a synchronisation set which blocks all the slow actions. Finding other decomposed components, and constructing the aggregated model, is achieved by removing this blocking synchronisation and allowing the current submodel to evolve just one step by a slow action. Note that the aggregated model does not have a representation at the MPA level, only as a Markov process.

Later work by Mertsiotakis [46,47], extends the class of models to which the technique can be applied by tackling the problem of *hybrid* components. These are sequential components which cannot be classified as either fast or slow since they enable both fast and slow actions. Mertsiotakis' solution is to extract the slow behaviour of the hybrid into a separate *shadow* component, making the original component passive with respect to these actions. In [47] it is shown that such a transformation preserves the behaviour of the hybrid component, and, since the equivalence relation is a congruence, the behaviour of the complete model. Once the transformation has been completed the original procedure of [37] can be applied.

6.2 Decision Free Processes

Another decomposed solution technique presented in Mertsiotakis' thesis allows throughput approximation for a class of MPA models termed *decision free processes*. This technique, developed with Silva in [48], is based on earlier work on throughput approximation in a class of SPN called *marked graphs* [43]. Essentially, the idea is to partition the model into components, typically two in the marked graph case. Decomposed models are then formed in which one component is fully represented while the other is reduced to a minimal form, usually consisting of a single transition. In addition to these two decomposed models there is also an extremely simple aggregated model, consisting of the two minimal forms linked appropriately. An iterative scheme is then used to find a solution to the model, the influence of one component on the other being represented in the decomposed model by the rate of the transition in the minimal form.

The MPA decision free process approach to throughput approximation [48,47] relies on the decomposition of a decision free process into three components, one of which acts as an intermediary between the other two. This component is distinguished as the *interface*. Note that the components do not necessarily correspond to sequential components (c.f. time scale decomposition). The decomposed Markov processes are generated from the consideration of the two possible (component, interface) pairs. In each case a reduced representation of the interface is used, corresponding to this component's view. In addition a basic skeleton is formed which corresponds to a greatly reduced version of the complete model, in which only the interface actions are carried out. Once this decomposition has been made, the algorithm follows the same general form as outlined above for the marked graph case.

Rather than a characterisation to recognise models of this form, this work relies on models having been constructed in the specified way. The class of decision free processes is defined via a reduced syntax, disallowing the choice combinator, +, and placing restrictions on where action types may appear within components. In particular each action type may be performed only once within any cycle of behaviour. This condition removes the possibility of implicit choice, when the action is to be carried out in synchronisation with another component. It is recognised that even working within this class the necessary structure, with an interface component acting as intermediary between two other components, may not be immediately apparent within the model. Therefore a series of possible transformations are defined, each of which is shown to be based on an equivalence relation which preserves some aspects of the model's behaviour. However, note that for two of the transformations the equivalence considers only functional, not temporal, behaviour. Thus it guarantees, for example, that no deadlock is introduced into the model but tells us nothing about the timing characteristics of the new model in relation to the old one.

6.3 Near-Independence

In [13], Ciardo and Trivedi present a decomposition technique for stochastic reward nets (a version of SPN with immediate transitions, inhibitor arcs and rewards) based on the notion of *near-independence*. Components are considered to be near-independent if they operate in parallel and only rarely interact. The basic idea is that near-independent components can be solved independently in conjunction with a graph model which represents the (limited) dependencies that do still exist between them. Several examples of canonical near-independent net structures are given in the paper, but in general recognising such structures in any given model, and whether necessary conditions on the graph model are met, rely on the expertise of the modeller.

Components are solved in isolation, as if they were independent, but their influence upon each other is approximated by the rate at which synchronisation can take place. This is estimated using the dependency graph. In general, fixed point iteration may be necessary in order to achieve a satisfactory solution of the complete model, depending on the structure of the graph.

In [7], Bohnenkamp and Haverkort suggest that this technique could be adapted for MPA models. This paper does not progress this directly in terms of an MPA language but does report some interesting experiments which investigate the feasibility of the approach. In the proposed approach the dependence between components is recognised as the actions on which they synchronise (synchronisation between delays is not allowed). In effect the behaviours of the near-independent components are serialised, first capturing the work which can be done until blocking occurs due to a synchronisation point and then the work necessary to achieve the synchronisation.

6.4 Decomposition via Synchronisation Points

In [8], Bohnenkamp and Haverkort develop the ideas from [7] in a slightly different direction. The approach still centres on the role of synchronisations between parallel components, but now the aim is to reformulate the underlying Markov process in terms of a set of semi-Markov processes. These semi-Markov processes are solved via their embedded Markov chains and evaluations of the distribution of the times between synchronisation points. Working within a MPA framework, they consider a class of models in which there is a fixed number of sequential processes composed in parallel, assuming each composition is subject to the same set of global synchronisation actions. Within this class of models their solution technique is exact with respect to throughputs and local steady state probabilities.

In the original MPA languages a delay is associated with each action representing its duration, resulting in a labelled transition system in which arcs are labelled by two types of information: action type and rate information. All the work already described in this chapter is based on such languages. However, Bohnenkamp and Haverkort use a language in which actions and time delays are treated separately. Several recent MPA languages [28,19] take this approach, which is also found in the timed process algebras. Here the transition system has two distinct transition relations: one representing instantaneous actions and the other representing the passing of (stochastic) time.

From the point of view of the work reported in [8], this simplifies somewhat as only actions are allowed to synchronise, and the authors do not need to be concerned about the meaning of synchronisation between two delays. The sequential components of the model are treated as the decomposed processes of the underlying Markov process. The aggregated model, the embedded Markov chain of a semi-Markov process, is constructed compositionally: a tensor expression is formed from the embedded Markov chain of the semi-Markov process corresponding to the synchronisation process of each sequential process algebra component. This EMC may have several disjoint components but the initial state of the process is used to ensure that only the "live" component is considered. The reader is referred to [8] and [9] for more details.

7 Future Prospects

Research over the last five years has clearly demonstrated that decomposed solution of MPA models is possible. Furthermore, it can be achieved by exploiting structures which are introduced at the process algebra level, to elucidate structures in the underlying Markov process. Of course, the class of models which can be recognised and handled by each of the current techniques, reported in this chapter, is somewhat limited. However the diversity of these techniques means that together they represent a substantial class of models.

In most of the work undertaken so far, the primary focus has been on characterising MPA models which, when the semantics of the language are applied

and the labelled transition system formed, generate Markov processes within the given class. The aim is to develop the characterisation as a set of syntactic conditions which can be tested without having to apply the semantics to the whole model, although investigation of the state space of individual components may be necessary. Moreover, these conditions should be sufficiently formal that they can be incorporated into one of the MPA tools, such as the PEPA Workbench [22], allowing the recognition of the structure to be automated. Note that in each case, the current characterisation is known to be incomplete in the sense that there are MPA models which give rise to Markov processes of the appropriate class which would not be recognised by the current conditions. Extending the characterisations is on-going work.

One way to extend a characterisation is to look for ways in which the syntactic conditions imposed on models can be relaxed whilst still generating an underlying Markov process that still satisfies the necessary conditions. Another way is to look for cases when a given MPA model, which does not apparently satisfy the characterisation, can be transformed into one which does. As with most process algebras, PEPA allows the same model to be expressed in alternative forms; for example, a pair of interacting components may be expressed as a single sequential component or a PEPA model $(P \underset{K}{\bowtie} R) \underset{L}{\bowtie} Q$ may be equivalent to $(P \parallel Q) \underset{K \cup L}{\bowtie} R$. In the case of the latter model, in the first form it is not a candidate for the product form over submodels while in the second form it is. Recent work at Edinburgh has been developing rewriting rules which allow a model to be systematically re-expressed towards the target of eliciting a quasi-reversible structure, if the underlying model has one.

When using term rewriting rules we are not altering the structure of the model (as represented by its labelled transition system), only the syntactic representation of it. In general, however, such non-intrusive manipulations will not be sufficient. The model will not belong to a class with efficient solution techniques. Thus another use of the known characterisations is as the target for model simplifications, in which a model that does not have the correct structure is modified until it does have one of the product form structures, and thus is more amenable to solution. The basic idea is that once a model has been constructed, with tool assistance, the modeller will be able to massage her model into one of the classes of models corresponding to a decomposed solution technique. This approach has been investigated in recent work by Hillston and Tomasik [61,40].

In [61] the authors study several variations on a PEPA model whose structure is close to that for the reversible class of product form solution. In each case some manipulation of the model is necessary in order to transform the model into a reversible one. The impact of the freedom afforded the modeller during these modifications (to choose activity rates etc.) are investigated together with the effect of the modifications on the performance measures of interest. In [40], a more formal approach to model simplification is taken. Whilst the motivation of this work is still transformation of a model to one which satisfies the criteria for reversibility, the focus is on defining a model simplification technique and assess-

ing its impact on both the steady state distribution and associated performance measures.

In general, interaction between components is the major barrier to exact decomposition (i.e. product form solution), although it holds the key to some of the aggregated decompositional techniques. There is scope to review the combinators for interaction which are offered by the MPA languages. As Clark's work on insensitive structures shows, it is possible to define a combinator which *guarantees* the independence of components from the point of view of the steady state distribution. Similarly *flow cooperation*, as defined in the characterisation of quasi-reversibility, captures an acceptable form of interaction. Further study needs to be made to see when models can be shown to be product form by construction, rather than by syntactic checking after construction.

The work that has been completed so far demonstrates that *automated* decomposed solution of MPA models is feasible. As such, it offers a solution, albeit partial, to the state space explosion problem. Moreover, in several cases this solution can already be applied transparently to the modeller [47]. This represents a significant step forwards for performance analysis using process algebras.

Acknowledgements

Jane Hillston's work is supported by the Engineering and Physical Sciences Research Council via the COMPA project (G/L10215). I would like to thank my colleagues who have contributed the interesting work reported throughout this chapter. I would like to apologise to those whose work I did not include: this was due to the limited space available.

References

1. A. Aldini, M. Bernardo, and R. Gorrieri. An algebraic model for evaluating the performance of an ATM switch with explicit rate marking. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 119–138. Prensas Universitarias de Zaragoza, September 1999.
2. H.H. Ammar and S.M. Rezaul Islam. Time Scale Decomposition of a Class of Generalized Stochastic Petri Net Models. *IEEE Transactions on Software Engineering*, 15(6):809–820, June 1989.
3. F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, Closed and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, April 1975.
4. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 202(1–2):1–54, July 1998.
5. M. Bhabuta, P.G. Harrison, and K. Kanani. Detecting reversibility in Markovian Process Algebra. In *Performance Engineering of Computer and Telecommunications Systems*, Liverpool John Moores University, September 1995. Springer-Verlag.
6. A. Blakemore and S. Tripathi. Automated Time Scale Decomposition of SPNs. In *Proc. of 5th International Workshop on Petri Nets and Performance Models (PNPM '93)*, Toulouse, 1993.

7. H. Bohnenkamp and B. Haverkort. Decomposition Methods for the Solution of Stochastic Process Algebra Models: a Proposal. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th Process Algebra and Performance Modelling Workshop*, 1997.
8. H. Bohnenkamp and B. Haverkort. Semi-Numerical Solution of Stochastic Process Algebra Models. In C. Priami, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1998.
9. H. Bohnenkamp and B. Haverkort. Stochastic event structures for the decomposition of stochastic process algebra models. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 119–138. Prensas Universitarias de Zaragoza, September 1999.
10. R.J. Boucherie. A Characterisation of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 20(7):536–544, July 1994.
11. R.J. Boucherie and M. Sereno. On the Traffic Equations of Batch Routing Queueing Networks and Stochastic Petri Nets. Technical report, European Research Consortium for Informatics and Mathematics, 1994.
12. P. Buchholz. Compositional Analysis of a Markovian Process Algebra. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.
13. G. Ciardo and K.S. Trivedi. A Decomposition Approach for Stochastic Petri Net Models. *Performance Evaluation*, 1992.
14. G. Clark. An Extended Weak Isomorphism for Model Simplification. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th Process Algebra and Performance Modelling Workshop*, 1997.
15. G. Clark. Stochastic Process Algebra Structure for Insensitivity. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 63–82. Prensas Universitarias de Zaragoza, September 1999.
16. G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models*. PhD thesis, LFCS, University of Edinburgh, 2000.
17. G. Clark and J. Hillston. Product form solution for an insensitive stochastic process algebra structure. *Performance Evaluation*, 2001. To appear.
18. P.J. Courtois. *Decomposability: Queueing and Computer System Applications*. ACM Series. Academic Press, New York, 1977.
19. P. D'Argenio, J-P. Katoen, and E. Brinksma. General Purpose Discrete Event Simulation Using Spades. In C. Priami, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1998.
20. S. Donatelli and M. Sereno. On the Product Form Solution for Stochastic Petri Nets. In *Application and Theory of Petri Nets*, pages 154–172. Springer Verlag, 1992.
21. J-M. Fourneau, L. Kloul, and F. Valois. Performance Modelling of Hierarchical Cellular Networks using PEPA. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 139–154. Prensas Universitarias de Zaragoza, September 1999.
22. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In G. Haring and G. Kotsis, editors, *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 794 of *LNCS*, pages 353–368. Springer-Verlag, 1994.

23. S. Gilmore, J. Hillston, R. Holton, and M. Rettelbach. Specifications in Stochastic Process Algebra for a Robot Control Problem. *International Journal of Production Research*, December 1995.
24. S. Gilmore, J. Hillston, and L. Recalde. Elementary structural analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh, 1997.
25. P. Harrison and J. Hillston. Exploiting Quasi-reversible Structures in Markovian Process Algebra Models. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.
26. W. Henderson, D. Lucic, and P.G. Taylor. A Net level Performance Analysis of Stochastic Petri Nets. *Journal of the Australian Mathematical Society, Series B*, 31:176–187, 1989.
27. W. Henderson and P.G. Taylor. Embedded Processes in Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 17(2):108 – 116, February 1991.
28. H. Hermanns and M. Rettelbach. Towards a Superset of Basic LOTOS for Performance Prediction. In M. Ribaudo, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1996.
29. H. Hermanns and M.L. Rettelbach. Syntax, Semantics, Equivalences and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.
30. U. Herzog. Formal description, time and performance analysis: A framework. Technical Report 15/90, IMMD VII, Friedrich-Alexander-Universität, Erlangen-Nürnberg, Germany, September 1990.
31. J. Hillston. The Nature of Synchronisation. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.
32. J. Hillston. Compositional Markovian Modelling Using a Process Algebra. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*. Kluwer, 1995.
33. J. Hillston. *A Compositional Approach to Performance Modelling*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.
34. J. Hillston. A Class of PEPA Models Exhibiting Product Form Solution. Technical Report ECS-LFCS-98-382, LFCS, Dept. of Computer Science, University of Edinburgh, February 1998.
35. J. Hillston, H. Hermanns, U. Herzog, V. Mertsiotakis, and M. Rettelbach. Integrating Qualitative and Quantitative Modelling with Stochastic Process Algebras. Technical report, IMMD VII, Universität Erlangen-Nürnberg, May 1994.
36. J. Hillston and L. Kloul. An Efficient Kronecker Representation for PEPA Models. Submitted for publication, 2000.
37. J. Hillston and V. Mertsiotakis. A Simple Time Scale Decomposition Technique for Stochastic Process Algebras. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.
38. J. Hillston and N. Thomas. A Syntactical Analysis of Reversible PEPA Models. In *Proc. of 6th Process Algebra and Performance Modelling Workshop*, Nice, France, September 1998. University of Verona.
39. J. Hillston and N. Thomas. Product Form Solution for a class of PEPA Models. *Performance Evaluation*, 35:171–192, 1999.
40. J. Hillston and J. Tomasik. Amalgamation of transition sequences in the PEPA formalism. In *Proc. of ICALP Workshops 2000 (PAPM)*. Carleton Scientific Press, 2000.
41. D.R.W. Holton. A PEPA Specification of an Industrial Production Cell. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

42. J.R. Jackson. Jobshop-like Queueing Systems. *Management Science*, 10:131–142, 1963.
43. H. Jungnitz. *Approximation Methods for Stochastic Petri Nets*. PhD thesis, Rensselaer Polytechnic Institute, May 1992.
44. F. Kelly. *Reversibility and Stochastic Processes*. Wiley, 1979.
45. A.A. Lazar and T.G. Robertazzi. Markovian Petri Net Protocols with Product Form Solution. *Performance Evaluation*, 12(1):67–77, January 1991.
46. V. Mertsiotakis. Time Scale Decomposition of Stochastic Process Algebra Models. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th Process Algebra and Performance Modelling Workshop*, 1997.
47. V. Mertsiotakis. *Approximate Analysis Methods for Stochastic Process Algebras*. PhD thesis, Universität Erlangen–Nürnberg, Martensstraße 3, 91058 Erlangen, September 1998.
48. V. Mertsiotakis and M. Silva. A Throughput Approximation Algorithm for Decision Free Processes. In M. Ribaudo, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1996.
49. V. Mertsiotakis and M. Silva. A Throughput Approximation Algorithm for Decision Free Processes. In M. Ribaudo, editor, *Proc. of 7th International Workshop on Petri Nets and Performance Models*, 1996.
50. I. Mitrani and N. Thomas. Routing Among Different Nodes Where Servers Break Down Without Losing Jobs. In F. Baccelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, pages 248–261. Springer, 1995.
51. I. Mitrani and P.E. Wright. Routing in the Presence of Breakdowns. *Performance Evaluation*, 20:151–164, 1994.
52. M.K. Molloy. Performance Analysis using Stochastic Petri Nets. *IEEE Transactions on Computers*, 31(9):913–917, September 1982.
53. B. Plateau. On the Stochastic Structure of Parallelism and Synchronisation Models for Distributed Algorithms. In *Proc. ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, 1985.
54. B. Plateau, J.M. Fourneau, and K.H. Lee. PEPS: A Package for Solving Complex Markov Models of Parallel Systems. In *Proc. of the 4th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 1988.
55. M.L. Rettelbach and M. Siegle. Compositional Minimal Semantics for the Stochastic Process Algebra TIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.
56. M. Sereno. Towards a Product Form Solution of Stochastic Process Algebras. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.
57. M. Sereno. *Performance Models for Discrete Event Systems with Synchronisations*, volume II, chapter Product form and Petri nets, pages 637–660. Kronos, 1998.
58. W.J. Stewart, K. Arif, and B. Plateau. The numerical solution of Stochastic Automata Networks. *European Journal of Operation Research*, 86(3):503–525, 1995.
59. N. Thomas and J. Bradley. Approximating variance in non-product form decomposed models. In *Proc. of ICALP Workshops 2000 (PAPM)*, pages 607–619. Carleton Scientific Press, 2000.
60. N. Thomas and S. Gilmore. Applying Quasi-Separability to Markovian Process Algebra. In *Proc. of 6th Process Algebra and Performance Modelling Workshop*, Nice, France, September 1998. University of Verona.
61. J. Tomasik and J. Hillston. Transforming PEPA Models to Obtain Product Form Bounds. Technical report, Division of Informatics, University of Edinburgh, 2000.

A Structured Operational Semantics for PEPA

Before stating the inference rules we must define the notion of *apparent rate*, which is used to define the rate of a shared activity. The apparent rate of an action type within a component represents the totally capacity of that component to carry out activities of that type when it is in its current state.

Definition 7 (Apparent Rate). *The* apparent rate *of action of type α in a component P, denoted $r_\alpha(P)$, is the sum of the rates of all activities of type α in $\mathcal{A}ct(P)$.*

1. $r_\alpha((\beta, r).P) = \begin{cases} r & \text{if } \beta = \alpha \\ 0 & \text{if } \beta \neq \alpha \end{cases}$
2. $r_\alpha(P + Q) = r_\alpha(P) + r_\alpha(Q)$
3. $r_\alpha(P/L) = \begin{cases} r_\alpha(P) & \text{if } \alpha \notin L \\ 0 & \text{if } \alpha \in L \end{cases}$
4. $r_\alpha(P \underset{L}{\bowtie} Q) = \begin{cases} \min(r_\alpha(P), r_\alpha(Q)) & \text{if } \alpha \in L \\ r_\alpha(P) + r_\alpha(Q) & \text{if } \alpha \notin L \end{cases}$

Note that an apparent rate may be unspecified: if P is defined as,

$$P \stackrel{\text{def}}{=} (\alpha, w_1\top).P_1 + (\alpha, w_2\top).P_2$$

then the apparent rate of α in component P is $r_\alpha(P) = (w_1 + w_2)\top$.

The apparent rate will be *undefined* for component expressions containing *unguarded* variables, i.e. variables which are not prefixed by an activity. Consequently we do not allow a component to be defined by such an expression.

Note that in cases of cooperation, the apparent rate of the shared activity will be the minimum of the apparent rates of the components involved, where $\min(\top, r) = r$ for all $r \in \mathbb{R}^+$. Thus we make an assumption that, in general, shared activities proceed at the rate of the slower of the two participating components. This is based on a notion that in general both components contribute some work to the shared activity. For a discussion of alternative assumptions see [31]. In the case of a passive action it is assumed that the corresponding component does not contribute to the work required to complete the shared activity.

The semantic rules, in the structured operational style of Plotkin, are presented here without comment; the interested reader is referred to [33] for more details. The rules are read as follows: if the transition(s) above the inference line can be inferred, then we can infer the transition below the line.

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} \ (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'} \ (\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \ (\alpha \in L), \qquad R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} \ (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} \ (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} \ (A \stackrel{\text{def}}{=} E)$$

Fig. 12. Operational semantics of PEPA

Stochastic Activity Networks: Formal Definitions and Concepts*

William H. Sanders[1] and John F. Meyer[2]

[1] Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL 61801, USA.
whs@crhc.uiuc.edu
[2] Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109, USA.
jfm@eecs.umich.edu

Abstract. Stochastic activity networks have been used since the mid-1980s for performance, dependability, and performability evaluation. They have been used as a modeling formalism in three modeling tools (METASAN, *UltraSAN*, and Möbius), and have been used to evaluate a wide range of systems. This chapter provides the formal definitions and basic concepts associated with SANs, explaining their behavior and their execution policy precisely.

1 Introduction

The development of model-based methods for evaluating computer systems and networks has as long a history as the systems themselves. When applied properly, these techniques can provide valuable insights into nonfunctional properties of a system, such as its performance, dependability, or performability. One approach in this regard has been the development of stochastic extensions to Petri nets. These extensions permit the representation of timeliness (real-time constraints) as well as parallelism in a stochastic setting. As models for performability evaluation [1], they also permit the representation of fault tolerance and degradable performance. Use of these nets was facilitated by the early recognition (see [2] and [3,4], for example) that, with an appropriate definition, their behavior could be represented as discrete-state Markov processes. Motivated by this representational power and solution capability, researchers sought to define particular

* This material is based upon work supported, in part, by the National Science Foundation under Grant No. 9975019 and by the Motorola Center for High-Availability System Validation at the University of Illinois (under the umbrella of the Motorola Communications Center). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of Motorola.

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 315–343, 2001.

variants of stochastic Petri nets well-suited to particular application needs or solution methods (DSPNs, GSPNs, SANs, and SRNs, for example).

One stochastic extension of these nets, known as "stochastic activity networks," was defined with the express purpose of facilitating unified performance/dependability (performability) evaluation as well as more traditional performance and dependability evaluation. In the time since their introduction, they have served as the basis for three modeling tools (METASAN [5], *UltraSAN* [6], and Möbius [7,8]), and have been used to evaluate a wide variety of systems. (See `www.crhc.uiuc.edu/PERFORM` for a partial list of references and information on how to get these tools.)

In order to be effective and generally applicable, a modeling scheme should have a formal basis that describes its primitives and behavior in an unambiguous way. A scheme must also be general enough to allow for easy representation of realistic systems, and formal enough to permit derivation of useful results. This chapter provides the formal definitions and concepts for stochastic activity networks (SANs), a variant of stochastic Petri nets. We first precisely define activity networks. Activity networks are the non-probabilistic model on which SANs are built, just as in a similar fashion, (un-timed) Petri nets provide the foundation for stochastic Petri nets. We then describe the execution of a SAN as a sequence of markings, activity completions, and case selections. With activity networks as a base, we then define stochastic activity networks, and describe precisely when a SAN's behavior is fully quantified in a probabilistic sense. When it is, we say that a SAN is *well-specified*. Finally, we provide basic algorithms for determining when a SAN is well-specified, using the structure of the net to do this efficiently.

While stochastic activity networks have been used since the mid-1980s, their formal definition appears only in dissertation form, and is not generally available. We hope that the definitions and concepts in this chapter will be more accessible, and aid other researchers who are developing and applying formal methods for stochastic evaluation of computer systems and networks.

2 Activity Networks

The desire to represent system characteristics of parallelism and timeliness, as well as fault tolerance and degradable performance, precipitated the development of general network-level performability models known as *stochastic activity networks* [9,10]. Stochastic activity networks are probabilistic extensions of "activity networks"; the nature of the extension is similar to the extension that constructs stochastic Petri nets from (classical) Petri nets.

2.1 Definitions

Informally (as in [10]), activity networks are generalized Petri nets with the following primitives:

- *activities*, which are of two kinds: *timed* activities and *instantaneous* activities. Each activity has a non-zero integral number of *cases* [1].
- *places*, as in Petri nets.
- *input gates,* each of which has a finite set of inputs and one output. Associated with each input gate are an n-ary computable predicate and an n-ary computable partial function over the set of natural numbers which are called the *enabling predicate* and the *input function*, respectively. The input function is defined for all values for which the enabling predicate is true.
- *output gates,* each of which has a finite set of outputs and one input. Associated with each output gate is an n-ary computable function on the set of natural numbers, called the *output function.*

Timed activities represent the activities of the modeled system whose durations impact the system's ability to perform. Instantaneous activities, on the other hand, represent system activities that, relative to the performance variable in question, are completed in a negligible amount of time. Cases associated with activities permit the realization of two types of spatial uncertainty. Uncertainty about which activities are enabled in a certain state is realized by cases associated with intervening instantaneous activities. Uncertainty about the next state assumed upon completion of a timed activity is realized by cases associated with that activity. Gates are introduced to permit greater flexibility in defining enabling and completion rules.

Before formally defining an activity network, it helps to define several related concepts in a more precise manner. Let P denote the set of all places of the network. If S is a set of places ($S \subseteq P$), a *marking of* S is a mapping $\mu : S \rightarrow I\!N$. Similarly, the set of *possible markings of* S is the set of functions $M_S = \{\mu \mid \mu : S \rightarrow I\!N\}$. With these definitions in mind, an *input gate* is defined to be a triple, (G, e, f), where $G \subseteq P$ is the set of *input places* associated with the gate, $e : M_G \rightarrow \{0, 1\}$ is the *enabling predicate* of the gate, and $f : M_G \rightarrow M_G$ is the *input function* of the gate. Similarly, an *output gate* is a pair, (G, f), where $G \subseteq P$ is the set of *output places* associated with the gate and $f : M_G \rightarrow M_G$ is an *output function* of the gate. One can then formally define an activity network in terms of allowable interconnections between these model primitives.

Definition 1 *An* activity network *(AN) is an eight-tuple*

$$AN = (P, A, I, O, \gamma, \tau, \iota, o)$$

where P is some finite set of places, A is a finite set of activities, I is a finite set of input gates, and O is a finite set of output gates. Furthermore, $\gamma : A \rightarrow I\!N^+$ specifies the number of cases for each activity, and $\tau : A \rightarrow \{Timed, Instantaneous\}$ specifies the type of each activity. The net structure is specified via the functions ι and o. $\iota : I \rightarrow A$ maps input gates to activities, while

[1] The term *case*, as used here, should not be confused with the notion of *cases* of elementary net systems [11]. Here the term *case* is used to denote a possible action that may be taken upon the completion of an event.

$o : O \rightarrow \{(a, c) | a \in A \text{ and } c \in \{1, 2, \ldots, \gamma(a)\}\}$ *maps output gates to cases of activities.*

Several implications of this definition are immediately apparent. First, each input or output gate is connected to a single activity. In addition, each input of an input gate or output of an output gate is connected to a unique place. In contrast to the definition in [10], different output gates and input gates of an activity may be connected to identical places, as has been done in practice. Ambiguity in the execution of the net is avoided by requiring that the marking obtained upon completion of each activity not depend on 1) the order of application of the input gate functions, or 2) the order of application of the output gate functions.

The following definitions aid in the discussion that follows.

Definition 2 *If* $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ *is an activity network and* $S, G \subseteq P$ *then*

1. *a mapping* $\mu : P \rightarrow \mathbb{N}$ *is a* marking of the network,
2. *for* $S \subseteq P$, $\mu_S : S \rightarrow \mathbb{N}$ *is the restriction of* μ *to places of* S *(i.e.* $\mu_S(p) = \mu(p), \forall p \in S$*),*
3. *an input gate* $g = (G, e, f)$ holds *in a marking* μ *if* $e(\mu_G) = 1$,
4. *an activity* a *is* enabled *in a marking* μ *if* g *holds for all* $g \in \iota^{-1}(a)$,
5. *a marking* μ *is* stable *if no instantaneous activities are enabled in* μ,
6. *the* input places of an activity a *consist of the set* $IP(a) = \{p \mid \exists\ (G, e, f) \in \iota^{-1}(a)$ *such that* $p \in G\}$, *and*
7. *the* output places of an activity a *consist of the set* $OP(a) = \{p \mid$ *for some* $c = 1, 2, \ldots, \gamma(a),\ \exists\ (G, f) \in o^{-1}(a, c)$ *such that* $p \in G\}$.

The marking of a network can alternatively be represented as a vector, in which each component of the vector is the number of tokens in a particular place. The correspondence of components of the vector to markings of places is done via some designated total ordering of P. For example, for a set of places $\{p_1, p_2, \ldots, p_n\} \subseteq P$ and marking vector $(n_1, n_2, \ldots, n_n)$, $\mu(p_1) = n_1, \mu(p_2) = n_2, \ldots, \mu(p_n) = n_n$, if $p_1 < p_2 < \ldots < p_n$. The functional notation for markings is more convenient for the development of theory, while the vector notation is useful for examples.

2.2 Graphical Representation

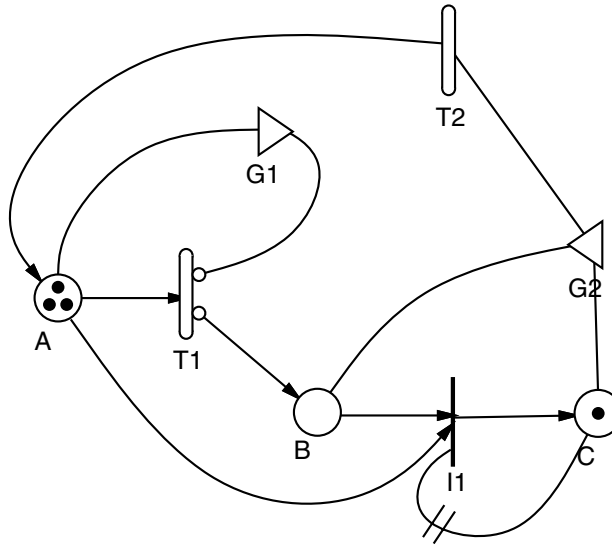
To aid in the modeling process, a graphical representation for activity networks is typically employed. In fact, for all but the smallest networks, specification via the tuple formulation presented in the definition is extremely cumbersome. Not only is the graphical representation more compact, but it also provides greater insight into the behavior of the network. For example, let i and j be natural numbers, and consider the following activity network:

$$
\begin{aligned}
P &= \{A, B, C\}, \text{ where } A < B < C \\
A &= \{T1, T2, I1\} \\
I &= \{GA1, GA2, GB, GC, G2\} \\
O &= \{AG, BG, CG, G1\} \\
\gamma &= \{(T1, 2),\ (T2, 1),\ (I1, 1)\} \\
\tau &= \{(T1, Timed),\ (T2, Timed),\ (I1, Instantaneous)\} \\
GA1 &= (\{A\},\ \{(i,1) \mid i > 0\} \cup \{(0,0)\},\ \{(i, i-1) \mid i > 0\} \cup \{(0,0)\}) \\
GA2 &= (\{A\},\ \{(i,1) \mid i > 0\} \cup \{(0,0)\},\ \{(i, i-1) \mid i > 0\} \cup \{(0,0)\}) \\
GB &= (\{B\},\ \{(i,1) \mid i > 0\} \cup \{(0,0)\},\ \{(i, i-1) \mid i > 0\} \cup \{(0,0)\}) \\
GC &= (\{C\},\ \{(i,0) \mid i > 0\} \cup \{(0,1)\},\ \{(i,i) \mid \forall i\}) \\
G2 &= (\{B, C\},\ \{((i,j),1) \mid i > 0 \text{ or } j > 0\} \cup \\
&\qquad \{(0,0),0)\},\ \{((i,j),(0,0)) \mid \forall i, j\}) \\
AG &= (\{A\},\ \{(i, i+1) \mid \forall i\}) \\
BG &= (\{B\},\ \{(i, i+1) \mid \forall i\}) \\
CG &= (\{C\},\ \{(i, i+1) \mid \forall i\}) \\
G1 &= (\{A, B\},\ \{((i,j),(i+2,j)) \mid i < 5 \text{ and } j < 5\} \cup \\
&\qquad \{((i,j),(i-1,j)) \mid j > 5 \text{ and } i = 0\} \cup \\
&\qquad \{((i,j),(i,j)) \mid j > 5 \text{ and } i \neq 0\}) \\
\iota &= \{(GA1, T1),\ (GA2, I1),\ (GB, I1),\ (GC, I1),\ (G2, T2)\} \\
o &= \{(AG, (T2, 1)),\ (BG, (T1, 2)),\ (CG, (I1, 1)),\ (G1, (T1, 1))\}
\end{aligned}
$$

Figure 1 depicts the graphical representation of this network.

One can immediately see the utility of a graphical representation. Here places are represented by circles (A, B, and C), as in Petri nets. Timed activities ($T1$ and $T2$) are represented as hollow ovals. Instantaneous activities ($I1$) are represented by solid bars. Cases associated with an activity are represented by small circles on one side of the activity (as on $T1$). An activity with only one case is represented with no circles on the output side (as on $T2$).

Gates are represented by triangles. $G2$ is an example of an input gate with 2 inputs. $G1$ is an example of an output gate with 2 outputs. Enabling predicates and functions for gates are typically given in tabular form. Three types of commonly used gates are given default (non-triangle) representations for ease of interpretation and to illustrate their similarity to classical Petri net primitives. In particular, an input gate with one input, enabling predicate $\{(i,1) \mid i > 0\} \cup \{(0,0)\}$, and function $\{(i, i-1) \mid i > 0\} \cup \{(0,0)\}$ (e.g., $GA1$) is represented as a directed line from its input to its output. Similarly, an output gate with one output and output function $\{(i, i+1) \mid \forall i\}$ (e.g., AG) is shown as a directed line from its input to its output. Finally, an input gate with one input, enabling predicate $\{(i,0) \mid i > 0\} \cup \{(0,1)\}$, and function $\{(i,i) \mid \forall i\}$ (e.g., GC) is shown as a directed line from its input to its output crossed by two short



Gate	Predicate	Function
G1	-	if (MARK(A) < 5 and MARK(B) < 5) then MARK(A) = MARK(A) + 2; else if (MARK(A) > 0) then MARK(A) = MARK(A) - 1;
G2	MARK(B)>0 or MARK(C)>0	MARK(B) = 0; MARK(C) = 0;

Fig. 1. Graphical Activity Network Representation

parallel lines. This type of input gate corresponds to an inhibitor arc in extended Petri nets. These shorthand notations for gates help the viewer understand the behavior of a network from its graphical representation.

2.3 Activity Network Behavior

The behavior of an activity network is a characterization of the possible completions of activities, selection of cases, and changes in markings. Specifically,

Definition 3 *An activity a* may complete *in a marking μ if*

1. *a is enabled in μ, and*
2. *if a is timed, no instantaneous activities are enabled in μ.*

This imposes two explicit priority classes on activities. We can now define the result of the completion of an activity and selection of a possible case. This is

made easier by expanding the domain and range of the gate functions to the complete network marking. Specifically, for an activity network with places P, a gate (of the network) with set of places G, and a function f, define the function $\tilde{f} : M_P \to M_P$ where if $\tilde{f}(\mu) = \mu'$ then

$$\mu'(p) = \begin{cases} f(\mu_G)(p) & \text{if } p \in G \\ \mu(p) & \text{otherwise.} \end{cases}$$

Using this notion, we can define an activity completion and case selection.

Definition 4 *Given an activity network $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ with activity a that may complete in μ, the* completion of activity a and choice of case c in μ yields

$$\mu' = \tilde{f}_{O_m}(\cdots \tilde{f}_{O_1}(\tilde{f}_{I_n}(\cdots \tilde{f}_{I_1}(\mu)\cdots))\cdots)$$

where $\iota^{-1}(a) = \{I_1, \ldots, I_n\}$ and $o^{-1}(a, c) = \{O_1, \ldots, O_m\}$.

While the gates in the two sets are numbered, there is no implied ordering of their application within a set, since the SAN definition does not specify an ordering among input gates or output gates. Output gate functions are applied after input gate functions, however. The notation $\mu \overset{a,c}{\to} \mu'$ is used to indicate that the completion of a and choice of c *yields* μ'. Furthermore, we say that marking μ' is *immediately reachable* from a marking μ if $\mu \overset{a,c}{\to} \mu'$ for some activity a and case c.

The set of reachable markings from a given marking can be defined directly in terms of the reflexive, transitive closure of the yields relation, which we denote as $\overset{*}{\to}$. Using this notation, the set of reachable markings from some marking can be defined as follows.

Definition 5 *The set of* reachable markings *of an activity network AN in a marking μ_0 is the set of markings $R(AN, \mu_0)$ where*

$$R(AN, \mu_0) = \{\mu \mid \mu_0 \overset{*}{\to} \mu\}.$$

Sets of "stable reachable markings" and "unstable reachable markings" of an activity network can then be defined in terms of its reachable markings. Specifically, the set of *stable reachable markings* of an activity network AN in an initial marking μ_0 is the set $SR(AN, \mu_0) \subseteq R(AN, \mu_0)$ of reachable markings of AN from μ_0 that are stable. Similarly, the set of *unstable reachable markings*, denoted $UR(AN, \mu_0)$, is the set of markings reachable from μ_0 that are not stable.

The behavior of an activity network can be described in terms of successive applications of the yields relation. Each application of the yields relation represents the completion of one of the one or more activities that may complete in the marking. Note that, unlike elementary net systems [11], the yields relation is defined only for single activities and that the concurrent completion of more than one activity is not considered. Each step in the evolution of the network is called a *configuration*, which, formally, is a marking-activity-case triple

$< \mu, a, c >$ where a is some activity with case c that may complete in μ. A *completion of a configuration* occurs when the activity associated with the configuration completes. The behavior of a network can thus be described in terms of possible sequences of configurations, more formally called *paths*.

Definition 6 *A* path *of an activity network, AN, with marking μ_0 is a sequence of configurations $\langle \mu_1, a_1, c_1 \rangle, \langle \mu_2, a_2, c_2 \rangle, \ldots, \langle \mu_n, a_n, c_n \rangle$ such that,*

1. $\mu_1 \in R(AN, \mu_0)$,
2. *for each pair of configurations* $\langle \mu_i, a_i, c_i \rangle$, $\langle \mu_{i+1}, a_{i+1}, c_{i+1} \rangle$ $(1 \leq i < n)$, $\mu_i \overset{a_i, c_i}{\rightarrow} \mu_{i+1}$, *and*
3. $\mu_n \overset{a_n, c_n}{\rightarrow} \mu'$ *for some marking* μ'.

Definition of several additional terms will aid in the discussion that follows. In particular, the *initial marking of a path* is the marking of the first configuration in the path. The *resulting marking of a path* is the marking that is reached upon completion of the last configuration in the path. A path is said to be *from μ to μ'* if μ is the initial marking of the path and μ' is the resulting marking of the path.

3 Stochastic Activity Networks

Activity networks are interesting in their own right, and several of their properties have been studied [12]. However, for the purpose of this chapter, they serve as a non-probabilistic base for a stochastic extension, called *stochastic activity networks*, that is used for performability evaluation. When they are used in this manner, care must be taken to insure that the probabilistic behavior of the stochastic extension is completely specified. Specifically, since we want to be able to ask questions regarding possible sequences of timed activity completions and intervening stable markings, we require that a stable marking eventually be reached after any sequence of consecutive instantaneous activity completions. Identification of situations in which this may occur is aided by the introduction of the notion of a *step*.

Definition 7 *Let AN be an activity network and s be a path of AN with initial marking μ_0. Then s is a step if:*

1. *the initial marking of s is stable, and*
2. *the markings of all other configurations of s are unstable.*

Note that the resulting marking of the step is not required to be stable. The set of markings that can be reached by completion of different steps from a single initial marking provides insight into the behavior of an activity network. To see this, let

$$S(\mu) = \{s \mid s \text{ is a step with initial marking } \mu\}$$

where μ is a stable reachable marking of the AN in question. Now, since there is only a finite number of steps of a given length from any marking μ, the

cardinality of $S(\mu)$ is $\aleph_0$ if and only if the length of steps in $S(\mu)$ increases without bound. Or, equivalently, since all of the activities except the first in a step are instantaneous, $|S(\mu)| = \aleph_0$ if and only if an unbounded number of instantaneous activities can complete without resulting in a stable marking. This leads us to the following definition of a "stabilizing" activity network. Formally,

Definition 8 *An activity network AN in a marking μ_0 is* stabilizing *if, for every $\mu \in SR(AN, \mu_0)$, the set $S(\mu)$ is finite.*

The following example illustrates the concept of stabilizing and non-stabilizing activity networks in a marking. Consider the activity network of Figure 2. If we denote its marking as a vector using the usual lexicographic

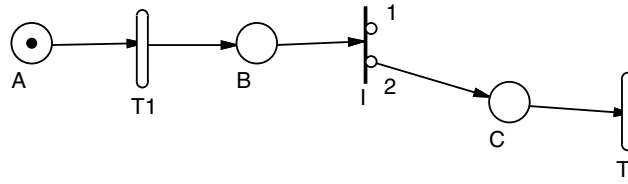


Fig. 2. A Stabilizing Activity Network

ordering of place names, then the set of steps associated with marking 100, i.e., the set $S(100)$, is

$$\{\langle 100, T1, 1\rangle\langle 010, I1, 1\rangle,\ \langle 100, T1, 1\rangle\langle 010, I1, 2\rangle\}.$$

Similarly, $S(001) = \{\langle 001, T2, 1\rangle\}$. These two markings are the only stable markings reachable from the pictured initial marking. Since both $S(100)$ and $S(001)$ are finite, the activity network is stabilizing. Now consider the activity network of Figure 3. For this network, $S(100)$ is the countably infinite set

$$\left\{ \begin{array}{l} \langle 100, T1, 1\rangle\langle 010, I1, 2\rangle, \\ \langle 100, T1, 1\rangle\langle 010, I1, 1\rangle, \\ \langle 100, T1, 1\rangle\langle 010, I1, 1\rangle\langle 010, I1, 1\rangle, \\ \langle 100, T1, 1\rangle\langle 010, I1, 1\rangle\langle 010, I1, 1\rangle\langle 010, I1, 1\rangle, \\ \vdots \end{array} \right\}.$$

Thus the activity network of Figure 3 is not stabilizing.

Generally, it is not decidable whether an activity network in a marking μ_0 is stabilizing. To see this, recall that it can be shown (see [13], for example) that extended Petri nets (Petri nets with inhibitor arcs) are equivalent, computationally, to Turing machines. The proof of this fact is by construction. Specifically,

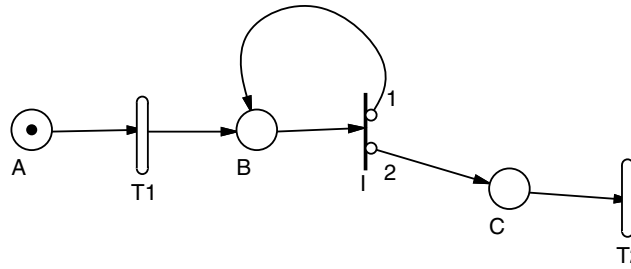


Fig. 3. An Activity Network that is Not Stabilizing

it can be shown that any register machine can be converted into an equivalent extended Petri net. For this reason, the languages generated by the net can be taken to be the set of sequences of transitions that lead to a reachable marking. Given this equivalence, it is evident that activity networks are equivalent to Turing machines, since every extended Petri net is an activity network (transitions map to activities, places to places, and arcs to gates). In the context of an activity network, the language generated can be taken to be the set of steps with initial marking μ_0 (i.e., $S(\mu_0)$). Thus the class of languages generated by the set of possible activity networks is coextensive with the class of recursively enumerable sets. Since, generally, it is not possible to decide whether a recursively enumerable set is finite [14], we have the following theorem.

Theorem 1 *It is not decidable whether an activity network in a marking μ_0 is stabilizing.*

There are, however, sufficient conditions by which the stabilizing property can be established, based on the structural properties and configuration of the instantaneous activities in the network. Identification of conditions is aided by the introduction of two properties of instantaneous activities. Specifically,

Definition 9 *An instantaneous activity is* self-disabling *if, given any reachable marking μ, it can only complete a finite number of times before any other activities complete.*

This definition allows us to identify activities that can only complete a bounded number of times before the markings of their input places change because of other activities. While this may be a difficult condition to check generally, it is easy to identify several frequently used activity-gate pairs that are self-disabling. For example, an activity with disjoint sets of input and output places and only default input gates (denoted by directed arcs) is self-disabling. In order to identify those activities that have no potentially unstabilizing interactions with other activities, we introduce the notion of a *cycle-free* instantaneous activity.

Definition 10 *An instantaneous activity* I_1 *is* cycle-free *if there does not exist a sequence of instantaneous activities* $I_1, I_2, \ldots, I_n$ *such that*

$$\begin{aligned} &OP(I_1) \cap IP(I_2) \neq \emptyset \ \wedge \\ &OP(I_2) \cap IP(I_3) \neq \emptyset \ \wedge \\ &\quad \vdots \\ &OP(I_n) \cap IP(I_1) \neq \emptyset. \end{aligned}$$

Informally, then, an instantaneous activity is cycle-free if there does not exist a sequence of instantaneous activities that, through their completions, can affect the original activity's input places. Note that this is a purely structural condition and that even if an activity is not cycle-free, the enabling predicates of the activities in the cycle may be such that the cycle of completions can never occur. Furthermore, because the number of activities in a network is finite by definition, an algorithm exists to determine whether an activity is cycle-free. These two concepts provide us with criteria to identify sufficient conditions for an activity network to be stabilizing. Intuitively, if every instantaneous activity is cycle-free and self-disabling, then no instantaneous activity can complete an unbounded number of times before the completion of a timed activity intervenes. More formally, we have the following theorem.

Theorem 2 *An activity network AN in a marking* μ *is stabilizing if every instantaneous activity of the network is cycle-free and self-disabling.*

Proof:

The proof is by contradiction. Suppose there exists an activity network AN with activities A that is not stabilizing in a marking μ, but in which every instantaneous activity of the network is cycle-free and self-disabling. Then, by definition, since AN is not stabilizing there exists an activity that can complete an unbounded number of times without resulting in a stable marking. A self-disabling activity can only complete an unbounded number of times without reaching a stable marking if another instantaneous activity changes the marking of one of its input places. But every instantaneous activity in A is cycle-free, so that can not occur. Thus an activity network in a marking μ is stabilizing if every instantaneous activity in the network is cycle-free and self-disabling. □

For an example of an activity network that satisfies the conditions of the above theorem, see Figure 4. First, note that all the instantaneous activities in the network are self-disabling, since they all have default input gates and disjoint input and output places. Secondly, note that all instantaneous activities are cycle-free. Thus by Theorem 2 this activity network is stabilizing. The stability of an activity network is an important necessary condition to insure that the probabilistic behavior of its stochastic extension is completely specified. Before the second necessary condition can be discussed, the definition of a stochastic activity network must be given.

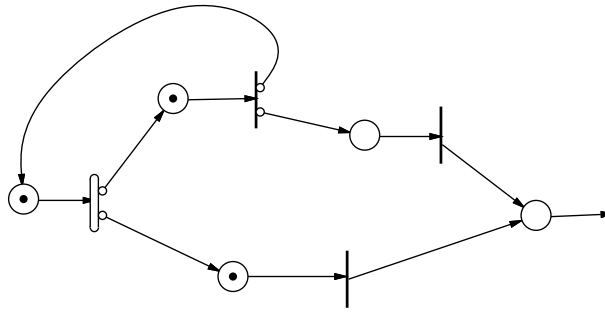


Fig. 4. A Second Stabilizing Activity Network

3.1 Definition of a Stochastic Activity Network

Given an activity network that is stabilizing in some specified initial marking, a stochastic activity network is formed by adjoining functions C, F, and G, where C specifies the probability distribution of case selections, F represents the probability distribution functions of activity delay times, and G describes the sets of "reactivation markings" for each possible marking. Formally,

Definition 11 *A* stochastic activity network *(SAN) is a five-tuple*

$$SAN = (AN, \mu_0, C, F, G)$$

where:

1. $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ *is an activity network.*
2. $\mu_0 \in M_P$ *is the* initial marking *and is a stable marking in which AN is stabilizing.*
3. C *is the* case distribution assignment*, an assignment of functions to activities such that for any activity a,* $C_a : M_{IP(a) \cup OP(a)} \times \{1, \ldots, \gamma(a)\} \rightarrow [0, 1]$. *Furthermore, for any activity a and marking* $\mu \in M_{IP(a) \cup OP(a)}$ *in which a is enabled,* $C_a(\mu, \cdot)$ *is a probability distribution called the* case distribution of a in μ.
4. F *is the* activity time distribution function assignment*, an assignment of continuous functions to timed activities such that for any timed activity a,* $F_a : M_P \times \mathbb{R} \rightarrow [0, 1]$. *Furthermore, for any stable marking* $\mu \in M_P$ *and timed activity a that is enabled in* μ, $F_a(\mu, \cdot)$ *is a continuous probability distribution function called the* activity time distribution function of a in μ*;* $F_a(\mu, \tau) = 0$ *if* $\tau \leq 0$.
5. G *is the* reactivation function assignment*, an assignment of functions to timed activities such that for any timed activity a,* $G_a : M_P \rightarrow \wp(M_P)$, *where* $\wp(M_P)$ *denotes the power set of* M_P. *Furthermore, for any stable marking* $\mu \in M_P$ *and timed activity a that is enabled in* μ, $G_a(\mu, \cdot)$ *is a set of markings called the* reactivation markings *of a in* μ.

Recall that an activity is enabled if all of its input gates hold. While this concept is sufficient to describe the behavior of an activity network, several additional terms are needed to describe the behavior of a stochastic activity network. Figure 5 aids in the description of these terms. In particular, since timed activities represent operations in a modeled system, events must be defined to denote the start and finish of these operations. The start of an operation is signaled by an *activation* of an activity. An activation of an activity will occur if 1) the activity becomes enabled (illustrated by the first time-line) or 2) the activity completes and is still enabled (illustrated by the second time-line). Some time after an activity is activated it will either *complete* (illustrated by the first time-line) or be *aborted* (illustrated by the third time-line). The activity will complete if it remains enabled throughout its activity time (which will be defined momentarily); otherwise it is aborted.

The activity time distribution function specifies (probabilistically) the *activity time* of an activity, i.e., the time between its activation and *completion*. Any continuous distribution (e.g., exponential or normal) is a legal activity time distribution, although the choice of distribution will affect the applicability of many solution methods. Both the distribution type and its parameters can depend on the global marking of the network at the activation time of the activity. Activity times are assumed to be mutually independent random variables.

Two other functions are associated with an activity network to form a SAN. In particular, the case distribution specifies (probabilistically) which case is to be chosen upon the completion of an activity. These probabilities can depend on the markings of the input and output places of the activity at its completion time. A reactivation function is also associated with each timed activity. This function specifies, for each marking, a set of *reactivation markings*. Given that an activity is activated in a specific marking, it is reactivated (i.e., activated again) whenever any marking in the set of reactivation markings for the activation marking is reached before the activity completes (as illustrated by the fourth time-line). Probabilistically, the reactivation of an activity is exactly the same as an activation; a new activity time distribution is selected based on the current marking. This provides a mechanism for restarting activities that have been activated, either with the same or a different distribution. This decision is made on a per-activity basis (based on the reactivation function) and, hence, is not a net-wide execution policy.

The definition of a stochastic activity network presented here differs from that presented earlier in [9] in three respects. First, probabilities associated with cases are required to depend only on input and output places of the associated activity. This requirement permits the development of more efficient algorithms to test whether the probabilistic behavior of a stochastic activity network is completely specified. Since any place in a network can be made to be an input or output place of any activity, this is not a restrictive requirement.

Second, the new definition requires that the probability distribution function associated with each timed activity in a possible marking be continuous. This requirement ensures that two timed activities do not complete at the same time,

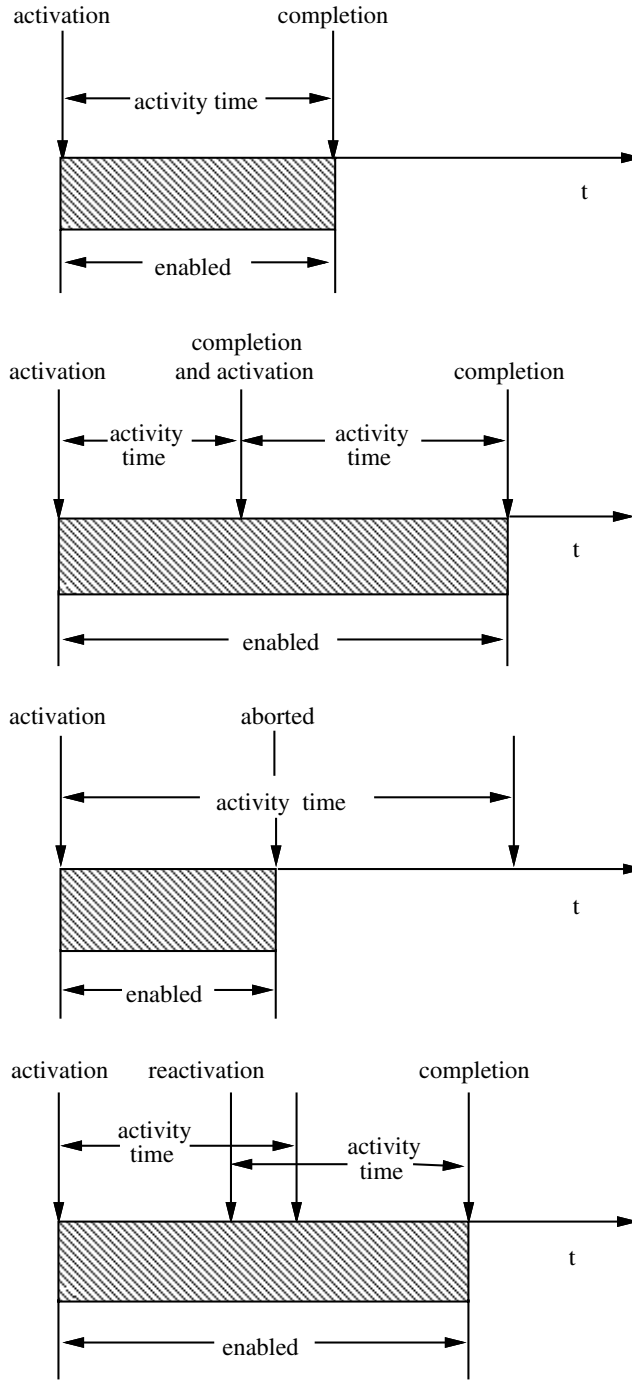


Fig. 5. Terms Related to the Execution of a Timed Activity

since the behavior when this occurs is not specified. This requirement is not overly strict for stochastic activity networks that are to be solved analytically, since the solution method usually places stricter constraints on the distributions. In the case of SANs that are to be solved via simulation, the ambiguity can be avoided by assigning an ordering to timed activities that may complete at the same time.

Third, the current definition does not require that the underlying activity network be "well-behaved" [9] in its initial marking. An activity network is said to be *well-behaved* in an initial marking μ if

1. No infinite sequence of instantaneous activities can complete in any marking reachable from marking μ, and
2. If a marking reachable from μ has more than one enabled instantaneous activity, then, from that marking, all possible sequences of reachable markings result in the same stable marking.

The requirement that the underlying activity network be well-behaved is more strict than the requirement that the underlying AN be stabilizing, and does lead to stochastic activity networks whose probabilistic behavior is completely specified. However, delaying the introduction of conditions of this type until after the stochastic extension is defined allows us to identify a larger class of networks whose probabilistic behavior is completely specified.

3.2 Stochastic Activity Network Behavior

Before discussing in detail how activity time distributions, case distributions, and reactivation functions determine an activity network's behavior, it helps to describe, informally, how a network executes in time. In particular, one can think of an execution of a SAN as a sequence of configurations, where for each configuration $\langle \mu, a, c \rangle$ the SAN was in marking μ, activity a completed, and case c was chosen. In any marking μ, the activity that completes is selected from the set of *active* activities in μ, i.e., the set of those activities that have been activated but have not yet completed or aborted. After each activity completion and case selection, the set of activities that are active is altered as follows. If the marking reached (as specified by the yields relation) is stable, then

1. the activity that completed is removed from the set of active activities,
2. activities that are no longer enabled in the reached marking are removed from the set of active activities,
3. activities for which the reached marking is a reactivation marking are removed from the set of active activities, and
4. activities that are enabled but are not in the set of active activities are placed in it (including those that were reactivated).

In contrast, if the marking reached is not stable, then timed activities (other than the one that just completed, if it is timed) are not added or deleted from the set. Instead,

1. the activity that completed is removed,
2. instantaneous activities that are no longer enabled in the reached marking are removed, and
3. instantaneous activities that are enabled but not in the set are added.

The choice of the activity to complete from the set of active activities is determined by the activity time distribution function of each activity in the set and the relative priority of timed and instantaneous activities (as specified by the definition of "may complete"). If there are one or more instantaneous activities in the set, one of them is chosen (non-deterministically) to complete. If there are none, the timed activity with the earliest completion time is selected (stochastically) based on the activity times of the set of active activities. Recall that the activity time distribution function defines the time between an activity's activation and completion. After the activity to complete is selected, a case of the activity is chosen based on its case distribution in the current marking, and a new marking is reached. The set of active activities is "initialized" at the start of an execution by adding to the set all those activities that are enabled in the initial marking. Note that, because choices between active instantaneous activities are made non-deterministically but not probabilistically, there may be networks for which these choices lead to behaviors that are not probabilistically specified. We now investigate conditions under which probabilistic behavior of the network is completely specified.

3.3 Well-Specified Stochastic Activity Networks

This section provides the basic concepts and ideas that define when a stochastic activity network's behavior is completely probabilistically specified. Since the time this material first appeared [15], further work has been done to develop algorithms that are tailored to specific reward variables [16], and are more efficient [17]. In addition, [18] presents a similar concept in the context of stochastic reward nets. These works present newer and more efficient algorithms to determine whether a net is well-specified; in contrast, this section focuses on the concept itself and how the structure of a SAN can be used to reduce the complexity of determining whether a SAN is well-specified.

Two types of nondeterminism can occur in stochastic activity networks: 1) uncertainty as to which activity will complete among the active activities, and 2) uncertainty as to which case of the activity that complete will be chosen. To aid in the discussion that follows, we will refer to a choice of the first type as an *activity choice* and a choice of the second type as a *case choice*.

In stochastic activity networks, case choices are quantified by the assignment of a case distribution to each activity. Activity choices are quantified partially by the assignment of an activity time distribution to each timed activity. However, the activity time distribution does not completely quantify this type of non-determinism, since the behavior is not defined if two activities have the same completion time. That would never occur for two timed activities, since all activity time distributions are continuous. It will occur, however, whenever

two or more instantaneous activities are enabled, since instantaneous activities complete in zero time. In this situation, the choice of which activity will complete next is non-deterministic and not quantified by either the activity time distribution function assignment or the case distribution assignment.

Since we are interested in possible sequences of timed activity completions and reached stable markings, we would like the probability distribution on the choice of the next stable marking to be the same regardless of any non-probabilistically quantified activity choices that have been made. To investigate this more formally, we introduce the notion of a *stable step.*

Definition 12 *Let $S = (AN, \mu_0, C, F, G)$ be a stochastic activity network and s be a step of AN with initial marking μ_0. Then, s is a* stable step *if the resulting marking of s is stable.*

A stable step can be thought of as a "jump" in the execution of a stochastic activity network that takes the network from one stable marking to another via the completion of a timed activity and zero or more instantaneous activities. There may be several steps with the same first marking and activity, but a different final marking. Accordingly, we define the "set of next stable markings" for a stable marking upon completion of an activity a as follows.

Definition 13 *Let $S = (AN, \mu_0, C, F, G)$ be a stochastic activity network and $\mu \in SR(AN, \mu_0)$. The set of* next stable markings *for S in μ upon completion of a is the set*

$$NS(\mu, a) = \left\{ \mu' \;\middle|\; \begin{array}{l} \exists \text{ a stable step } s \text{ from } \mu \text{ to } \mu' \text{ such that} \\ a \text{ is the activity of the first configuration of } s \end{array} \right\}.$$

This set allows us to focus our attention on those stable markings that may be reached from a particular stable marking by completion of a specific timed activity. In order to insure that the probability distribution over the next stable markings is invariant over activity choices between instantaneous activities, we must define a measure on stable steps that captures the probability that a stable step will be taken given that a set of activity choices is made in a manner such that the step is possible. The case construct allows us to define this probability. Specifically,

Definition 14 *Let $S = (AN, \mu_0, C, F, G)$ be a SAN and let s be a path of S such that $s = \langle \mu^1, a^1, c^1 \rangle \langle \mu^2, a^2, c^2 \rangle \cdots \langle \mu^n, a^n, c^n \rangle$. Then,* the probability of *s, denoted $Pr(s)$, is*

$$C_{a^1}(\mu^1_{IP(a^1) \cup OP(a^1)}, c^1) \times C_{a^2}(\mu^2_{IP(a^2) \cup OP(a^2)}, c^2) \ldots \times C_{a^n}(\mu^n_{IP(a^n) \cup OP(a^n)}, c^n)$$

where $\times$ is taken to be normal multiplication on the set of real numbers.

This function defines the probability that a stable step will be taken given that a set of activity choices was made such that the step may occur. Since we want to insure that the probability distribution over the next stable markings

upon completion of a timed activity is invariant over possible sets of activity choices, we consider the set of steps from some stable marking μ to a stable marking $\mu' \in NS(\mu, a)$, by completion of a timed activity a that may complete in μ. Formally, let

$$P^a_{\mu,\mu'} = \{ s \mid s \text{ is a stable step from } \mu \text{ to } \mu' \text{ with first activity } a \}.$$

Different stable steps in this set can arise from different sets of activity choices. In order to specify the set of stable steps from one marking to another upon completion of some timed activity for a single set of activity choices, it helps to define a relation relating stable steps that can occur under a single set of activity choices. Specifically, define the relation R on $P^a_{\mu,\mu'}$ to be

$$R = \left\{ (s, s') \;\middle|\; \begin{array}{l} \text{for every configuration } \langle \mu, a, c \rangle \text{ in } s \text{ and configuration} \\ \langle \mu', a', c' \rangle \text{ in } s' \text{ such that } \mu = \mu',\ a = a' \end{array} \right\}.$$

Thus, two stable steps are related if, for every marking they share in common, the same activity choice is made. Note that while R is not an equivalence relation, it is a compatibility relation (i.e., it is reflexive and symmetric). While a compatibility relation does not necessarily define a partition of a set, it does define a covering of a set, by the *maximal compatibility classes* of the relation. Recall that (as in [19]) a subset $C \subseteq P^a_{\mu,\mu'}$ is called a *maximal compatibility class* if every element of C is related to every other element of C and no element of $P^a_{\mu,\mu'} - C$ is related to all the elements of C. Each maximal compatibility class contains stable steps that correspond to a single set of activity choices. More specifically, note that all stable steps in C correspond to steps from μ to μ' by completion of timed activity a under some set of activity choices. The probability that μ' is reached from μ by completion of a, given that a particular set of activity choices has been made, is thus the sum of the probabilities of all stable steps in C, i.e.,

$$P(C) = \sum_{s \in C} Pr(s).$$

All activity choices within stable steps correspond to choices between active instantaneous activities and, hence, are not probabilistically specified. Therefore, for a stochastic activity network to be completely probabilistically specified, $P(C)$ must be the same for all maximal compatibility classes C. To express this precisely, we introduce the notion of a *well-specified stochastic activity network*.

Definition 15 *A stochastic activity network $S = (AN, \mu_0, C, F, G)$ is* well-specified *if, for every marking $\mu \in SR(AN, \mu_0)$, each activity a that may complete in μ, and all $\mu' \in NS(\mu, a)$, $P(C)$ is identical for all maximal compatibility classes C of R defined on $P^a_{\mu,\mu'}$.*

The above definition identifies a class of networks whose behavior is completely specified, probabilistically, with respect to all notions of state that we intend to consider.

It is interesting to compare the notion just presented to the "well-behaved" notion used previously. In particular, one can show that every activity network well-behaved in some marking μ_0 is well-specified for any choice of activity time distributions, reactivation functions, and case probabilities. We state this fact as a theorem.

Theorem 3 *If an activity network is well-behaved in a marking μ_0, then it is well-specified for any choice of activity time distributions, reactivation functions, and case distributions.*

Proof:

Suppose an activity network AN is well-behaved in a marking μ_0. Then, for every marking μ reachable from μ_0, no infinite sequence of activities can complete in μ. Thus AN is stabilizing in μ_0. Augment AN with arbitrary activity distributions, reactivation functions, case distributions, and initial marking μ_0 to form a SAN. Now, recall that this SAN is well-specified if for every stable marking μ reachable from μ_0, each activity a that may complete in μ, and each $\mu' \in NS(\mu, a)$, $P(C)$ is identical for all maximal compatibility classes C or R defined on $P^a_{\mu,\mu'}$. Consider an arbitrary stable marking μ reachable from μ_0 and activity a that may complete in μ. Since S is well-behaved, one of three situations may arise upon completion of a in μ.

In the first situation, all markings in the set of next possible markings are stable. There is thus only one maximal compatibility class for this marking and activity, and the criterion of Definition 15 is satisfied.

In the second situation, at least one marking in the set of next possible markings is unstable, and all possible unstable markings that may be reached before another stable marking is reached have at most one instantaneous activity enabled. Then, as in the first situation, there is only one maximal compatibility class for this marking and activity, and therefore the criterion of Definition 15 is satisfied.

In the third situation, at least one marking in the set of next possible markings is unstable, and some unstable marking that may be reached before another stable marking is reached that has two or more instantaneous activities enabled. But, since S is well-behaved, all possible sequences of markings reachable from that marking will result in the same next stable marking. Thus, while there may be more than one compatibility class for the marking and activity, they all result in the same single marking with a probability of one, and hence, $P(C)$ is identical for all maximal compatibility classes. Again, the criterion of Definition 15 is satisfied for this marking and activity.

Since the criterion of the definition is satisfied for each possible situation for every reachable marking and activity that may complete in the marking, the SAN is well-specified for any choice of activity time distributions, reactivations, and case distributions. □

It should be noted, however, that the converse of 3 is not a theorem, and hence that well-specified SANs are more general than well-behaved SANs. To see this, consider the stochastic activity network of Figure 6. This SAN is well-

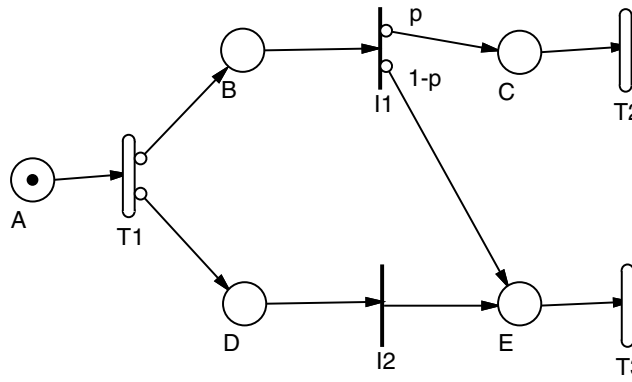


Fig. 6. A Well-Specified, but not Well-Behaved, Stochastic Activity Network

specified in the pictured marking, since the activity choice that is made after completion of the enabled timed activity does not affect the distribution of the next stable markings. It is not well-behaved, though, since two instantaneous activities may be enabled and more than one stable marking can be reached from the current marking. Recall that in order for an activity network to be well-behaved, whenever a reached marking has two or more instantaneous activities enabled, all possible sequences of reachable markings must result in the same stable marking.

Any algorithm to determine whether a given stochastic activity network is well-specified must check that the probability distribution over each next stable markings does not depend on the set of activity choices that is made. This condition can be checked using techniques developed to find the set of all maximal compatibles [20]. The following algorithm checks this for a particular stable marking and timed activity.

Algorithm 1 *(Determines whether the next stable marking probability distribution is invariant over possible sets of activity choices for a stable marking μ and activity a that can complete in μ, and if it is, computes this distribution.)*

Compute the set of all stable steps for which the initial marking is μ and timed activity is a.
Group the set of stable steps computed above according to the resulting marking of each step. The subset containing the stable steps from μ to μ' by completion of timed activity a is denoted by $P^a_{\mu,\mu'}$.
For each subset $P^a_{\mu,\mu'}$:
    Construct the set of maximal compatibles of R on $P^a_{\mu,\mu'}$.
    Compute $P(C)$ for each maximal compatible C.
    If $P(C)$ is not identical for all compatibles C, then
        Signal SAN is not well-specified and abort algorithm.
Next $P^a_{\mu,\mu'}$.

Return next stable marking probability distribution for marking and activity.

For convenience, we denote this distribution by the function $h_{\mu,a} : NS(\mu, a) \rightarrow [0, 1]$, where for $\mu' \in NS(\mu, a)$, $h_{\mu,a}(\mu')$ is the probability that μ' will be reached given that the SAN is in μ and a completes.

The following example illustrates the use of Algorithm 1. Specifically, consider the stochastic activity network of Figure 7. Here the case distribution for each

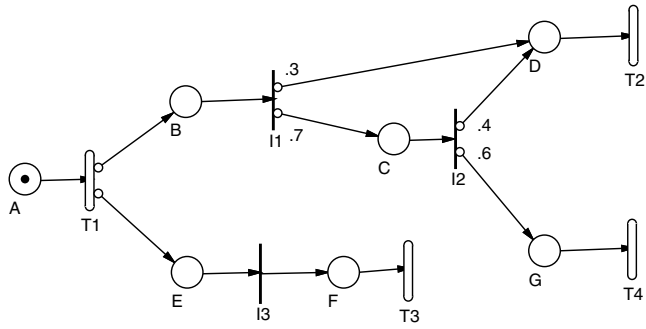


Fig. 7. A Well-Specified Stochastic Activity Network

activity is denoted by the number next to each case for the activity. In addition, markings are denoted by vectors, assuming the usual lexicographic ordering of place names. With these facts in mind, we will show (using Algorithm 1) that the next stable marking probability distribution is invariant over all possible sets of activity choices for the initial marking 1000000 (lexicographic ordering on place names) and timed activity $T1$. The algorithm first computes the set of all stable steps, which is shown in Figure 8. These steps are then used to determine

$$\left\{\begin{array}{l}
\langle 1000000, T1, 1\rangle\langle 0100100, I1, 1\rangle\langle 0001100, I3, 1\rangle, \\
\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I2, 1\rangle\langle 0001100, I3, 1\rangle, \\
\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I2, 2\rangle\langle 0000101, I3, 1\rangle, \\
\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I3, 1\rangle\langle 0010010, I2, 1\rangle, \\
\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I3, 1\rangle\langle 0010010, I2, 2\rangle, \\
\langle 1000000, T1, 1\rangle\langle 0100100, I3, 1\rangle\langle 0100010, I1, 1\rangle, \\
\langle 1000000, T1, 1\rangle\langle 0100100, I3, 1\rangle\langle 0100010, I1, 2\rangle\langle 0010010, I2, 1\rangle, \\
\langle 1000000, T1, 1\rangle\langle 0100100, I3, 1\rangle\langle 0100010, I1, 2\rangle\langle 0010010, I2, 2\rangle
\end{array}\right\}$$

Fig. 8. Set of Stable Steps for the Stochastic Activity Network of Figure 2.7 from Marking 10000000 by Completion of Activity $T1$.

the set of possible next stable markings ($NS(1000000, T1)$), which is found to be $\{0001010, 0000011\}$. The set of stable steps is then split into two subsets, according to their resulting markings. These two subsets serve as input to the portion of the algorithm that computes sets of maximal compatibles, checks to see that the probability measure is invariant over all possible compatibles, and computes the next stable marking probabilities. The algorithm then computes the set of maximal compatibles corresponding to the resulting marking 0001010; the set consist of the three elements

$$C_1 = \{\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I2, 1\rangle\langle 0001100, I3, 1\rangle, \\ \langle 1000000, T1, 1\rangle\langle 0100100, I1, 1\rangle\langle 0001100, I3, 1\rangle\},$$

$$C_2 = \{\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I3, 1\rangle\langle 0010010, I2, 1\rangle, \\ \langle 1000000, T1, 1\rangle\langle 0100100, I1, 1\rangle\langle 0001100, I3, 1\rangle\}, \text{ and}$$

$$C_3 = \{\langle 1000000, T1, 1\rangle\langle 0100100, I3, 1\rangle\langle 0100010, I1, 2\rangle\langle 0010010, I2, 1\rangle, \\ \langle 1000000, T1, 1\rangle\langle 0100100, I3, 1\rangle\langle 0100010, I1, 1\rangle\}.$$

$P(C)$ is then computed for each maximal compatible C, and $P(C_1) = P(C_2) = P(C_3) = .58$. Similarly, computing the set of maximal compatibles of stable steps with resulting marking 0000011, we obtain:

$$C_1 = \{\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I2, 2\rangle\langle 0000101, I3, 1\rangle\}$$
$$C_2 = \{\langle 1000000, T1, 1\rangle\langle 0100100, I1, 2\rangle\langle 0010100, I3, 1\rangle\langle 0010010, I2, 2\rangle\}$$
$$C_3 = \{\langle 1000000, T1, 1\rangle\langle 0100100, I3, 1\rangle\langle 0100010, I1, 2\rangle\langle 0010010, I2, 2\rangle\}$$

For these compatibles, $P(C_1) = P(C_2) = P(C_3) = .42$. Since the probability measure is the same for all the maximal compatibles in a set, for both sets, the next stable marking probability distribution is invariant over the set of possible activity choices for this SAN, starting marking, and timed activity.

By definition, then, a stochastic activity network is well-specified if this distribution is invariant for all stable reachable markings and activities that may complete in these markings. In cases in which the set of stable reachable markings is finite, we define the following algorithm which determines whether a SAN is well-specified.

Algorithm 2 *(Determines whether a SAN with a finite reachability set is well-specified and computes next stable marking probability distributions)*

For each $\mu \in SR(AN, \mu_0)$ and activity a that may complete in μ:
Determine whether the next stable marking probability distribution is invariant over possible sets of activity choices for this choice of μ and a.
If distribution is not invariant for this μ and a, then
Signal SAN is not well-specified and abort algorithm.
Next $\mu \in SR(AN, \mu_0)$ and activity a that may complete in μ.
Signal stochastic activity network is well-specified.

Note that this algorithm is simply an iterative application of Algorithm 1. While Algorithm 1 suffices to determine whether the next stable marking probability distribution is invariant for a particular marking μ and activity a, its performance can be improved upon if information concerning the structure of the network is used. This technique makes use of the concept of *dependent instantaneous activities*. Specifically,

Definition 16 *Let I_1 and I_2 be instantaneous activities of some activity network. Then I_1 and I_2 are* dependent *if*

$$(IP(I_1) \cup OP(I_1)) \cap (IP(I_2) \cup OP(I_2)) \neq \emptyset.$$

Informally, then, two instantaneous activities are dependent if they have common input or output places. Two activities that are dependent can affect each other by changing the markings of each other's input or output places. In order to identify instantaneous activities that can affect each other through a sequence of completions, we look at the transitive closure of a relation based on the above definition. Specifically, let DEP denote a relation on the set of instantaneous activities of a SAN such that $I_1\ DEP\ I_2$ if I_1 and I_2 are dependent. Furthermore, let $\overset{*}{DEP}$ denote the transitive closure of DEP. It is easy to see that $\overset{*}{DEP}$ is an equivalence relation; thus, it partitions the set of instantaneous activities. The blocks of the partition are sets of activities whose order of completion may affect the probability distribution of the next stable markings. On the other hand, pairs of activities from different blocks cannot affect each other by completing (since activities can only change the markings of their input or output places, and case probabilities depend only on input and output places). This suggests that steps within each subnetwork defined by activities in each block can be considered individually and combined to determine the total probability for a possible next stable marking.

In order to explore this idea in more detail, we define the notion of an instantaneous subnetwork of a SAN constructed from a set of activities.

Definition 17 *Given a stochastic activity network $S = (AN, \mu_0, C, F, G)$, underlying activity network $AN = (P, A, I, O, \gamma, \tau, \iota, o)$, and set of instantaneous activities $A' \subseteq A$, the* instantaneous subnetwork of S with respect to A' *is a structure (M', μ_0', C') where*

1. *$M' = (P', A', I', O', \gamma', \tau', \iota', o')$ is an activity network with*
   (a) *$P' = \{p \mid p \in P$ and $p \in IP(a) \cup OP(a)$ for some $a \in A'\}$,*
   (b) *A' is some specified set of instantaneous activities,*
   (c) *$I' = \{g \mid g \in I$ and $g \in \iota^{-1}(a)$ for some $a \in A'\}$,*
   (d) *$O' = \{g \mid g \in O$ and $g \in o^{-1}(a, c)$ for some $a \in A'$ and $c = 1, 2, \ldots, \gamma(a)\}$, and*
   (e) *γ', τ', ι', and o' are the functions γ, τ, ι, and o, respectively, restricted to P', A', I', and O'.*
2. *$\mu_0' = \mu_0$ restricted to places P', and*
3. *C' is the function C restricted to A'.*

While (M', mu'_0, C') does not fit the definition of a stochastic activity network precisely since the initial marking is not stable, it does provide us with a network made up of instantaneous activities in which all the case probabilities are specified. The revised algorithm presumes that such a subnetwork is constructed for each set of activities corresponding to a block of the partition defined by $\overset{*}{DEP}$. By the nature of $\overset{*}{DEP}$, these subnetworks do not interact with one another. This fact is exploited in the revised algorithm presented below.

Unlike Algorithm 1, which computes the set of all stable steps for the given marking and activity immediately, the revised algorithm accomplishes the same goal in two smaller steps. First, it computes the set of "next possible markings" for the given starting marking and activity. The set of next possible markings is the set of markings that can be reached by one application of the yields relation, i.e., for a given marking μ and activity a

$$NP(\mu, a) = \{\mu' \mid \mu \overset{a,c}{\rightarrow} \mu' \text{ for some } a \in A \text{ and } c \in \{1, \ldots, \gamma(a)\}\}.$$

Each of these markings can be either stable or unstable. If a marking is stable, then it is a next stable marking for the specified starting marking and activity. Furthermore, its probability of occurrence is just the sum of the probabilities of all cases that lead to that marking. Since only probabilistically specified activity choices were made in reaching the marking, the network is well-specified. A more complicated situation exists for each unstable marking in $NP(\mu, a)$.

These markings are those unstable markings that can be reached after one application of the yields relation and, hence, represent situations where one or more instantaneous activities must be completed to reach possible next stable markings. To determine these markings, consider the instantaneous subnetworks previously constructed. First, note that although the sets of places defined by each subnetwork are disjoint, they may not partition the set of places of the entire network, since there may be places that are connected only to timed activities. The markings of these places will not change by completion of instantaneous activities and hence will remain the same in all next stable markings of the network reached from this marking. In the algorithm that follows, the marking of the places connected only to timed activities is denoted by μ'_s, for each $\mu' \in NP(\mu, a)$. Similarly, the initial (unstable) markings of each of the n subnetworks are denoted by $\mu'_1, \mu'_2, \ldots, \mu'_n$, respectively, for each $\mu' \in NP(\mu, a)$. Now, since the markings of places of different subnetworks are independent of each other, sets of next stable markings can be computed for each subnetwork independently and be combined to obtain "global" next stable markings for the entire network.

Computation of the local next stable markings, and the subsequent check that the probabilities these markings are invariant over possible sets of activity choices for each subnetwork, is done in a manner similar to that of Algorithm 1, except that the initial marking of each of the possible paths to a stable marking is not itself stable. Since these paths are suffixes of stable steps, we call them *partial stable steps*. A partial stable step is a stable step without the initial configuration. Except for this difference, the invariant check and computation of probabilities are done exactly as in Algorithm 1. In the algorithm presented

below, for each subnetwork i, the set of (subnetwork) next stable markings is denoted by NS_i and the probability that subnetwork marking $\mu''_{i,j}$ will be reached from subnetwork marking μ'_i is denoted by $\hat{h}_{\mu'_i}(\mu''_{i,j})$.

After the possible next stable markings and probabilities of these markings have been computed for each subnetwork, they are combined to construct next stable markings and probabilities for the entire network. Possible next stable markings for the entire network are constructed by forming all possible combinations of μ'_s's and subnetwork next stable markings. Each marking constructed that way is denoted by the concatenation of its constituent subnetwork markings together with μ'_s. The probability of each of these global markings is then computed as the product of the probabilities of each of the constituent markings. Since each global marking could also be reached in other ways (i.e., from another $\mu' \in NP(\mu', a)$), the computed probability obtained in each way is summed to obtain the total probability for this next stable marking.

A more precise description of the algorithm is the following.

Algorithm 3 *(Uses concept of instantaneous subnetworks to determine, given a stable marking μ and activity a that can complete in μ, whether the next stable marking probability distribution is invariant over possible sets of activity choices, and if it is, computes this distribution.)*

Let $NS(\mu, a)$ equal the null set.
Let $h_{\mu,a} = 0$.
Compute $NP(\mu, a)$.
For each $\mu' \in NP(\mu, a)$:
  Let $\bar{h}_{\mu,a}(\mu') = \sum_{c \text{ such that } \mu \xrightarrow{a,c} \mu'} C_a(\mu_{IP(a) \cup OP(a)}, c)$.
  If μ' is stable then
    Add μ' to $NS(\mu, a)$.
    Let $h_{\mu,a}(\mu') = \bar{h}_{\mu,a}(\mu')$.
  else
    For each instantaneous subnetwork i, $i = 1$ to n:
      Restrict μ' to places of the subnetwork (this is denoted by μ'_i).
      Compute the set of all partial stable steps of the subnetwork with initial marking μ'_i.
      Compute the set of resulting stable markings from the set of partial stable steps. Label these k_i markings $\mu''_{i,1}$, $\mu''_{i,2}$, ..., μ''_{i,k_i}.
      Group the set of partial stable steps computed above according to their resulting markings. $P_{\mu'_i,\mu''_{i,j}}$ denotes the subset containing partial stable steps from μ'_i to $\mu''_{i,j}$.
      For each $P_{\mu'_i,\mu''_{i,j}}$, $j = 1$ to k_i:
        Construct the set of maximal compatibles of R on $P_{\mu'_i,\mu''_{i,j}}$.
        Compute $P(C)$ for each maximal compatible C.
        If $P(C)$ is not identical for all compatibles C then
          Signal SAN is not well-specified and abort algorithm.

else
Add $\mu''_{i,j}$ *to* $NS_i(\mu'_i)$.
Let $\hat{h}_{\mu'_i}(\mu''_{i,j}) = P(C)$.
Next j.
Next i.
{* *Now form global next stable marking from subnetwork results* *}
For $j_1 = 1$ *to* k_1
For $j_2 = j_1$ *to* k_2
$\ddots$
For $j_n = j_{n-1}$ *to* k_n

Add $\mu'_{1,j_1}\mu'_{2,j_2}\cdots\mu'_{n,j_n}\mu'_s$ *to* $NS(\mu, a)$.
Let $h_{\mu,a}(\mu'_{1,j_1}\mu'_{2,j_2}\cdots\mu'_{n,j_n}\mu'_s) =$
$h_{\mu,a}(\mu'_{1,j_1}\mu'_{2,j_2}\cdots\mu'_{n,j_n}\mu'_s) + \bar{h}_{\mu,a}(\mu')\prod_{i=1}$ *to* $_n\, \hat{h}_{\mu'_i}(\mu'_{i,j_i})$.
Next j_n.
Next j_2.

Next j_1.
Next $\mu' \in NP(\mu, a)$.
Return next stable marking probability distribution for marking and activity.

This algorithm has significant advantages over Algorithm 1 for systems that have several instantaneous subnetworks, and reduces to Algorithm 1 when there is a single subnetwork. To illustrate this, consider again the stochastic activity network of Figure 7, which was shown using Algorithm 1 to be well-specified. Figure 2.9 depicts this activity network with the instantaneous subnetworks outlined. As shown by Algorithm 3, each of these subnetworks can be analyzed

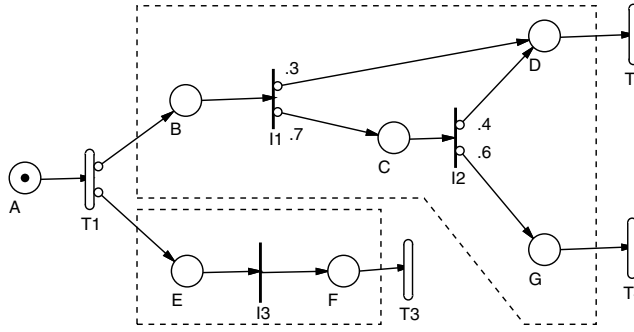


Fig. 9. A Well-Specified Stochastic Activity Network with Instantaneous Subnetworks Noted

separately to check that the network is well-specified and to compute the next stable state probability distribution. In order to do this, we must first compute the set of next possible markings, which in this case is {0100100}. The probability of this marking, denoted $\bar{h}_{1000000,T1}(0100100)$, is 1. Now, since the marking is not stable, the set of next stable states and their probabilities must be computed for each instantaneous subnetwork. For the first subnetwork, which contains $I1$ and $I2$, the set of partial stable steps is

$$\left\{ \begin{array}{l} \langle 1000, I1, 1\rangle, \\ \langle 1000, I1, 2\rangle\langle 0100, I2, 1\rangle, \\ \langle 1000, I1, 2\rangle\langle 0100, I2, 2\rangle \end{array} \right\}$$

and the set of possible next stable markings, NS_1, is $\{0010, 0001\}$. There is one maximal compatible corresponding to each possible next stable marking. The first,

$$C = \{\langle 1000, I1, 1\rangle, \langle 1000, I1, 2\rangle, \langle 0100, I2, 1\rangle\},$$

has probability $P(C) = .58$, and hence $\hat{h}_{0010} = .58$. The second, corresponding to the stable marking 0001, is

$$C = \{\langle 1000, I1, 2\rangle\langle 0100, I2, 2\rangle\}$$

and has probability $\hat{h}_{1000}(0001) = .42$. The computations for the second subnetwork are even simpler. For it, the set of partial stable steps is the singleton set

$$\{\langle 10, I3, 1\rangle\}$$

where the set of possible stable markings, NS_2, is $\{01\}$ and $\hat{h}_{10}(01) = 1$.

Since the next stable state probabilities are invariant for each subnetwork, they are invariant for the entire network. The global next stable marking probabilities are computed by forming possible combinations of the local next stable markings. When this is done, the set of next stable markings is found to be $NS(1000000) = \{0001010, 0000011\}$ with probabilities $h_{1000000,T1}(0001010) = .58$ and $h_{1000000,T1}(0000011) = .42$. This result matches that computed previously using Algorithm 1.

4 Conclusion

This chapter has presented a formal definition of activity networks and stochastic activity networks, formally described their behavior, and specified conditions which describe when their behavior is completely probabilistically defined. By providing formal definitions of these nets, we were able to precisely define when they can be used for evaluation. Other publications have defined the framework for defining reward variables on SANs [21] and the stochastic process representations of the behavior of SANs [22], when these conditions are met.

References

1. J. F. Meyer, "On evaluating the performability of degradable computing systems," *IEEE Transactions on Computers*, vol. C-22, no. 10, pp. 720–731, Aug. 1980.
2. M. K. Molloy, "Performance analysis using stochastic Petri nets," *IEEE Transactions on Computers*, vol. C-31, pp. 913–917, 1982.
3. G. Florin and S. Natkin, "Les Reseaux de Petri Stochastiques," *Technique et Science Informatiques*, vol. 4, no. 1, pp. 143–160, 1985.
4. B. Beyaert, G. Florin, P. Lonc, and S. Natkin, "Evaluation of computer systems dependability using stochastic Petri nets," in *Proc. 11th Int. Symp. Fault-Tolerant Computing (FTCS-11)*, Portland, Maine, USA, 1981, pp. 79–81, IEEE Computer Society Press.
5. W. H. Sanders and J. F. Meyer, "METASAN: A Performability Evaluation Tool Based on Stochastic Activity Networks," in *Proceedings of the IEEE-ACM Fall Joint Computer Conference*, Dallas, TX, November 1986, pp. 807–816.
6. W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko, "The *UltraSAN* Modeling Environment," *Performance Evaluation*, vol. 24, no. 1, pp. 89–115, October-November 1995.
7. D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders, "Möbius: An extensible tool for performance and dependability modeling," in *Computer Performance Evaluation: Modelling Techniques and Tools: Proceedings of the 11th International Conference, TOOLS 2000*, B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith, Eds. vol. 1786 of *Lecture Notes in Computer Science*, pp. 332–336, Berlin, Springer-Verlag.
8. W. H. Sanders, "Integrated frameworks for multi-level and multi-formalism modeling," in *Proceedings of PNPM'99: 8th International Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, September 1999, pp. 2–9.
9. J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: structure, behavior and application," *Proc. International Workshop on Timed Petri Nets*, pp. 106–115, 1985.
10. A. Movaghar and J. F. Meyer, "Performability modeling with stochastic activity networks," in *Proc. 1984 Real-Time Systems Symposium*, Austin, TX, December 1984.
11. P. S. Thiagarajan, "Elementary net systems," in *Petri nets: central models and their properties*, W. Brauer, Ed. 1986, vol. 254 of *Lecture Notes in Computer Science*, pp. 26–59, Berlin, Springer-Verlag.
12. A. Movaghar, *Performability modeling with stochastic activity networks*, Ph.D. thesis, University of Michigan, 1985.
13. J. L. Peterson, *Petri Net Theory and the Modelling of Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
14. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, N. Reading, MA, 1980.
15. W. H. Sanders, *Construction and solution of performability models based on stochastic activity networks*, Ph.D. thesis, University of Michigan, 1988.
16. M. A. Qureshi, W. H. Sanders, A. P. A. van Moorsel, and R. German, "Algorithms for the generation of state-level representations of stochastic activity networks with general reward structures," *IEEE Transactions on Software Engineering*, vol. 22, no. 9, pp. 603–614, Sept. 1996.
17. D. D. Deavours and W. H. Sanders, "An efficient well-specified check," in *Proceedings of PNPM'99: 8th International Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, September 1999, pp. 124–133.

18. G. Ciardo and R. Zijal, "Well-defined stochastic Petri nets," in *Proceedings of the Fourth International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'96)*, San Jose, California, Feb. 1996, pp. 278–284.
19. J. P. Tremblay and R. Manohar, *Discrete Mathematical Structures with Applications to Computer Science*, McGraw-Hill, New York, 1975.
20. Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1978.
21. W. H. Sanders and J. F. Meyer, "A unified approach for specifying measures of performance, dependability, and performability," *Dependable Computing and Fault Tolerant Systems*, vol. 4, pp. 215–237, 1991.
22. W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 1, pp. 25–36, Jan. 1991.

Distributed and Structured Analysis Approaches to Study Large and Complex Systems*

Gianfranco Ciardo

Department of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
ciardo@cs.wm.edu

Abstract. Both the logic and the stochastic analysis of discrete-state systems are hindered by the combinatorial growth of the state space underlying a high-level model. In this work, we consider two orthogonal approaches to cope with this "state-space explosion". Distributed algorithms that make use of the processors and memory overall available on a network of N workstations can manage models with state spaces approximately N times larger than what is possible on a single workstation. A second approach, constituting a fundamental paradigm shift, is instead based on decision diagrams and related implicit data structures that efficiently encode the state space or the transition rate matrix of a model, provided that it has some structure to guide its decomposition; with these implicit methods, enormous sets can be managed efficiently, but the numerical solution of the stochastic model, if desired, is still a bottleneck, as it requires vectors of the size of the state space.

1 Introduction

As digital systems are becoming ubiquitous and their complexity is steadily growing, it is increasingly important to be able to study their logical and timing behavior. While direct observation, testing, and measurement are sometimes feasible, they are normally very expensive, and, by definition, can only be employed after the system has been built. Discrete-state models are then an attractive, general, and inexpensive alternative that provides a way to study a system at various levels of detail, even when it is still just a concept in the designer's mind. However, the discrete nature of the system implies that its "state" is the collection of the states of each of its components, resulting in the well-known "combinatorial explosion" of the state space, which poses a formidable analysis challenge even for today's powerful computers.

In this work, we consider techniques to cope with this state-space explosion problem, focusing in particular on models that have an underlying continuous-time Markov chain (CTMC). Our presentation is split into three main portions.

* This work was supported by the National Aeronautics and Space Administration under NASA Grant NAG-1-2168.

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 344–374, 2001.

Sec. 3 introduces the main concepts related to the underlying state space and CTMC, and their traditional solution methods. Sec. 4 discusses distributed analysis algorithms that can be used to spread the memory and time requirements over a network of N workstations, thus are able to cope with state spaces approximately N time larger. Sec. 5 moves instead to what we call "implicit" techniques which require much less than linear memory to store the state space and the CTMC. In the former case, this also results in enormous time savings, while, in the latter, the approach usually involves a memory-time tradeoff. A unified treatment of these two aspects shows some of the commonalities and research challenges. Finally, in Sec. 6 we briefly conclude with our thoughts about fruitful directions of future research.

2 Notation

We use italic letters to indicate scalars (e.g., a), calligraphic letters to indicate sets (e.g., $\mathcal{A}$), lower case boldface Roman or Greek letters to indicate row vectors (e.g., $\mathbf{a}$, $\boldsymbol{\alpha}$), and upper case boldface Roman letters to indicate matrices (e.g., $\mathbf{A}$). Vector and matrix elements are indicated using square brackets (e.g., $\mathbf{a}[1]$, $\mathbf{A}[1,2]$), and we extend the notation to sub-vectors or sub-matrices by allowing sets of indices to be used instead of single indices (e.g., $\mathbf{a}[\mathcal{A}]$, $\mathbf{A}[\mathcal{A},\mathcal{B}]$).

We indicate with $\text{diag}(\mathbf{a})$, $\mathbf{I}_n$, and $\mathbf{0}$ the diagonal matrix with vector $\mathbf{a}$ on its main diagonal, the identity matrix of size $n \times n$ (n is omitted if clear from the context), and a row vector of 0's of the appropriate dimension, respectively.

We use subscripts to indicate event indices (e.g., $\mathbf{R}_e$); in the discussion of distributed approaches, we also use subscripts, but in square brackets, for process indices (e.g., $\mathcal{S}_{[n]}$) while, in the discussion on implicit methods, we use subscripts, without square brackets, for sub-model indices (e.g., $\mathcal{S}_k$); if both sub-model and event indices are present, they appear in that order (e.g., $\mathbf{W}_{k,e}$). We consistently number and use submodel indices "going down" from K to 1, never up; the reader should keep this in mind when operators such as Kronecker product and sum are used, since these are not commutative.

States are denoted in boldface lower case letters, just as vectors, to stress that they are somehow structured, that is, they are a collection of sub-states.

3 High-Level Models and Traditional Solution Methods

Rather than discussing analysis techniques for a particular formalism such as Petri nets [34] or process algebras [3], we introduce instead a general framework for discrete-state models. This is preferable since nothing in our discussion is tied to the particular formalism chosen. However, we will sometimes make specific references to Petri nets, since we regard them as quite graphically intuitive (or perhaps simply because we are most familiar with them!). For more information on Petri nets and their stochastic extensions, see G. Balbo, this volume.

ExploreExplicitSequential : set of *state*

Build and return the state space $\mathcal{S}$.

declare $\mathcal{S}, \mathcal{U}$: set of *state*;
declare $\mathbf{i}, \mathbf{j}$: *state*;

1. $\mathcal{S} \Leftarrow \emptyset$; • initialize the explored states
2. $\mathcal{U} \Leftarrow \{\mathbf{s}^{initial}\}$; • initialize the unexplored states to the initial state
3. while $\exists\ \mathbf{i} \in \mathcal{U}$ do • there are still states to explore
4. for each $\mathbf{j} \in \mathcal{N}(\mathbf{i})$ do
5. if $\mathbf{j} \notin \mathcal{U} \cup \mathcal{S}$ then • found a new state $\mathbf{j}$
6. $\mathcal{U} \Leftarrow \mathcal{U} \cup \{\mathbf{j}\}$;
7. end if;
8. end for;
9. $\mathcal{U} \Leftarrow \mathcal{U} \setminus \{\mathbf{i}\}$; • move $\mathbf{i}$ from unexplored ...
10. $\mathcal{S} \Leftarrow \mathcal{S} \cup \{\mathbf{i}\}$; • ... to explored
11. end while;
12. return $\mathcal{S}$;

Fig. 1. An explicit sequential algorithm to generate the state space.

3.1 State-Space Generation

We consider discrete-state models having a finite underlying state space $\mathcal{S}$. Any such high-level model must describe the following objects in a compact way:

- $\widehat{\mathcal{S}}$, the set of *potential states*, describing the "type" of the system states.
- $\mathbf{s}^{initial} \in \widehat{\mathcal{S}}$, the *initial state* of the system.
- $\mathcal{N} : \widehat{\mathcal{S}} \to 2^{\widehat{\mathcal{S}}}$, the *next-state function*, describing which states can be reached from a given state in a single step, or transition.

In many formalisms, the next-state function $\mathcal{N}$ is expressed as $\mathcal{N} = \bigcup_{e \in \mathcal{E}} \mathcal{N}_e$, where $\mathcal{E}$ is a finite set of *events*, $\mathcal{N}_e$ is the next-state function associated with event e, and the union operator is naturally meant to be applied to the values of the involved functions, that is, $\mathcal{N}(\mathbf{i}) = \bigcup_{e \in \mathcal{E}} \mathcal{N}_e(\mathbf{i})$ Then, $\mathcal{N}_e(\mathbf{i})$ is the set of states the system can enter when event e occurs, or *fires*, in state $\mathbf{i}$. Note that, with a single function, we encompass not only the concept of *next state*, but also that of *enabling*, since event e is enabled in $\mathbf{i}$ iff $|\mathcal{N}_e(\mathbf{i})| > 0$, and of *non-determinism*, since the effect of event e is non-deterministic in state $\mathbf{i}$ iff $|\mathcal{N}_e(\mathbf{i})| > 1$. In addition, of course, a model exhibits non-determinism if multiple events are enabled in a state and there is no a priori way to choose among them.

Given a high-level model, we can then start from $\mathbf{s}^{initial}$ and build $\mathcal{S}$ using the *explicit state-space generation* algorithm whose pseudo-code is listed in Fig. 1. In other words, $\mathcal{S} \subseteq \widehat{\mathcal{S}}$ is the smallest set that contains the initial system state $\mathbf{s}^{initial}$ and is closed with respect to $\mathcal{N}$:

$$\mathcal{S} = \{\mathbf{s}^{initial}\} \cup \mathcal{N}(\mathbf{s}^{initial}) \cup \mathcal{N}(\mathcal{N}(\mathbf{s}^{initial})) \cup \cdots = \mathcal{N}^*(\mathbf{s}^{initial}),$$

where we have extended the function $\mathcal{N}$ to allow a set of states to be its argument, and "*" denotes reflexive and transitive closure.

The runtime and memory requirements for *ExploreExplicitSequential* depend on the data structure used to store $\mathcal{U}$ and $\mathcal{S}$. The key operations are determining the new states reachable from a given state $\mathbf{i}$ (statement 4), performed $|\mathcal{S}|$ times, and searching whether each of them is already in $\mathcal{U}$ or $\mathcal{S}$ (statement 5), performed $|\mathcal{A}| = \sum_{\mathbf{i} \in \mathcal{S}} |\mathcal{N}(\mathbf{i})|$ times, where $\mathcal{A}$ are the possible state-to-state transitions in the system (for a Petri net, this is the cardinality of the arc set in its *reachability graph*). If we assume that the cost of computing $\mathcal{N}(\mathbf{i})$ is proportional to $|\mathcal{N}(\mathbf{i})|$, that states are stored as individual entities, and that a balanced search tree is used to store $\mathcal{U}$ and $\mathcal{S}$ (see [19,28] for a detailed discussion of alternative techniques), the overall time complexity of *ExploreExplicitSequential* is $O(|\mathcal{A}| \cdot \log |\mathcal{S}|)$ and its memory complexity is $O(|\mathcal{S}|)$.

Once $\mathcal{S}$ has been explored and stored, we can query it for the presence or absence of states satisfying a certain condition, or we can ask more complex questions that involve the existence of *paths* in the reachability graph. In the former case, the query requires at most to scan each state once, so it can be answered in $O(|\mathcal{S}|)$ time. In the latter case, the complexity depends on the particular query; a recent trend is to employ *temporal logic* to specify the properties to be checked, resulting in the so-called *model checking* [22]. In our discussion, we assume that state-space generation is a goal in itself; this is the case either if we are only interested in reachability-type queries, such as "can the system reach a deadlock" (see, for example, the model checking tool Uppaal [30]), or if state-space generation is just an intermediate step in our ultimate goal, performing a stochastic analysis.

Before concluding this section, we observe that we often need to associate a *unique integer index* to a given a state $\mathbf{i}$. We can do so with a mapping

$$\Psi : \mathcal{S} \rightarrow \{0, \ldots, |\mathcal{S}| - 1\} \qquad \text{satisfying} \qquad \Psi(\mathbf{i}) = \Psi(\mathbf{j}) \Rightarrow \mathbf{i} = \mathbf{j}.$$

While any mapping will do, two possibilities are mostly used in practice. In traditional solution methods, it might be convenient to define $\Psi(\mathbf{i}) = i$ iff $\mathbf{i}$ was the i^{th} state "discovered" by *ExploreExplicitSequential*; hence, in particular, $\Psi(\mathbf{s}^{initial}) = 0$; this is a very useful choice, especially since it allows us to know the index of a state even before completing the exploration of $\mathcal{S}$. In the structured methods we consider, $\Psi(\mathbf{i})$ is instead the position of $\mathbf{i}$ in $\mathcal{S}$ according to lexicographic order; this mapping is well-defined only once the exploration process is complete, and requires a comparison operator defined over $\mathcal{S}$ (of course, such an operator is already required by traditional methods that use a search tree to store $\mathcal{S}$). Since Ψ is a bijection, its inverse Ψ^{-1} exists. Also, we will at times use a set of states $\mathcal{A}$ as a parameter to Ψ, with the obvious meaning $\Psi(\mathcal{A}) = \{\Psi(\mathbf{i}) : \mathbf{i} \in \mathcal{A}\}$.

3.2 Markov Chain Specification and Solution

If we are interested in the timing or probabilistic behavior of the system, the *logic* specification of the previous section must be augmented with *stochastic* information associated with each state-to-state transition. In our discussion, we limit

ourselves to the case of models having an underlying CTMC[1], so our high-level model needs to specify $Rate(\mathbf{i},\mathbf{j})$, the rate at which the system, when in state $\mathbf{i}$, transitions to state $\mathbf{j}$, for each reachable state $\mathbf{i} \in \mathcal{S}$ and each $\mathbf{j} \in \mathcal{N}(\mathbf{i})$; by definition, $Rate(\mathbf{i},\mathbf{j}) = 0$ iff $\mathbf{j} \notin \mathcal{N}(\mathbf{i})$. For more information on CTMCs, see "Markov Chains for Performance and Dependability Evaluation", by B. Haverkort, this volume.

Just as for the next-state function, this rate is normally expressed in a per-event fashion: $Rate(\mathbf{i},\mathbf{j}) = \sum_{e\in\mathcal{E}} Rate_e(\mathbf{i},\mathbf{j})$, where $Rate_e(\mathbf{i},\mathbf{j})$ is the rate at which event e leads from state $\mathbf{i}$ to state $\mathbf{j} \in \mathcal{N}_e(\mathbf{i})$.

We then define the *transition rate matrix* $\mathbf{R}$ of the underlying CTMC as

$$\mathbf{R} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|} \quad \text{where} \quad \mathbf{R}[i,j] = Rate(\mathbf{i},\mathbf{j}) \quad \text{and} \quad i = \Psi(\mathbf{i}),\ j = \Psi(\mathbf{j}).$$

The entries of $\mathbf{R}$ can be computed and stored during the for-loop iterations in *ExploreExplicitSequential*. Alternatively, it is often more efficient to generate the state space $\mathcal{S}$ first while, at the same time, just counting the number $\eta(\mathbf{R})$ of nonzero entries in $\mathbf{R}$, essentially the number of arcs in $\mathcal{A}$. Then, we can allocate an efficient *row-pointer column-index*[2] data structure [40] that requires only $|\mathcal{S}| + 1 + \eta(\mathbf{R})$ integers and $\eta(\mathbf{R})$ floating point numbers. A second state-space generation pass can then be used to fill this data structure with the actual values.

Given matrix $\mathbf{R}$, we can define the *holding-time vector* $\mathbf{h}$ expressing the expected time the CTMC spends in each state before making a transition:

$$\mathbf{h}[i] = \left(\sum_{0\le j<|\mathcal{S}|} \mathbf{R}[i,j] \right)^{-1},$$

and the *infinitesimal generator matrix* $\mathbf{Q}$, which equals $\mathbf{R}$ except in its diagonal entries[3], defined as

$$\mathbf{Q}[i,i] = -\sum_{0\le i<|\mathcal{S}|, j\neq i} \mathbf{R}[i,j] = \mathbf{R}[i,i] - \mathbf{h}[i]^{-1}.$$

Then, the numerical stationary solution of an *ergodic* CTMC involves the computation of the vector $\boldsymbol{\pi} \in \mathbb{R}^{|\mathcal{S}|}$ of *stationary state probabilities*, solution of

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0} \qquad \text{subject to} \qquad \sum_{0\le i<|\mathcal{S}|} \boldsymbol{\pi}[i] = 1. \tag{1}$$

[1] An analogous discussion is valid for the discrete-time Markovian case, while more general stochastic processes present many subtle difficulties.

[2] Or, rather, *column-pointer row-index*, as we often need only by-column access to $\mathbf{R}$.

[3] Unlike most definitions of CTMCs, our allows for the existence of *self-transitions* $\mathbf{R}[i,i]$ in the transition rate matrix. These are useless from a stochastic point of view, since they can be eliminated without changing the meaning and stochastic behavior of the model. However, we explicitly consider them because, when using the structured approaches of Sect. 5.2, the resulting CTMC might exhibit them. Of course, these are apparent only when considering $\mathbf{R}$ and $\mathbf{h}$ separately, while $\mathbf{Q}$ does not reflect their presence, since they cancel out.

If the CTMC is instead *absorbing*, we might instead be interested in computing the vector $\boldsymbol{\sigma} \in \mathbb{R}^{|\mathcal{T}|}$ of *expected state sojourn times until absorption*, solution of $\boldsymbol{\sigma}\mathbf{Q}[\Psi(\mathcal{T}), \Psi(\mathcal{T})] = -\boldsymbol{\pi}(0)[\Psi(\mathcal{T})]$, where $\mathcal{T}$ is the set of transient states and $\boldsymbol{\pi}(0)$ is the *initial probability vector*, whose entries, in our case, would be all zero except for a one in correspondence to $\mathbf{s}^{initial}$, i.e., $\boldsymbol{\pi}(0)[\Psi(\mathbf{s}^{initial})] = 1$. Regardless of the nature of the CTMC, we might instead want to compute a transient solution, that is, the probability vector at time t or the time spent in each state up to time t. For simplicity, we do not consider these other types of analysis here, since we focus on the data structures and discrete algorithms that allows us to tackle models with large state spaces, but we stress that our discussion applies also to absorbing Markov chains and transient analysis.

Many numerical algorithms are available for the solution of the linear homogeneous system of Eq. 1. In practice, $\mathcal{S}$ is very large and $\mathbf{R}$ is very sparse, i.e., only a small portion of its entries are nonzero. Thus, *iterative* methods are preferred, where successive approximations of the exact solution $\boldsymbol{\pi}$ are computed starting from an initial guess, without modifying $\mathbf{R}$, whose sparsity is preserved. We now describe the iteration performed by some popular solution methods:

- **Power**: $\boldsymbol{\pi}^{new} \Leftarrow \boldsymbol{\pi}^{old}(\mathbf{I} + \mathbf{Q}h^*)$, where h^* is a value slightly smaller than the smallest expected sojourn time in any state, $h^* < \min_{0 \leq i < |\mathcal{S}|}\{\mathbf{h}[i]\}$. Element-wise, this corresponds to:

  for $j = 0$ to $|\mathcal{S}| - 1$ do
    $a \Leftarrow \boldsymbol{\pi}^{old}[j]$;
    for $i = 0$ to $|\mathcal{S}| - 1$ do
      $a \Leftarrow a + \boldsymbol{\pi}^{old}[i]\mathbf{Q}[i, j]h^*$;
    end for;
    $\boldsymbol{\pi}^{new}[j] \Leftarrow a$;
  end for;

  • a is a high-precision accumulator

  The Power method is guaranteed to converge in theory, but it is often extremely slow in practice.
- **Jacobi**: $\boldsymbol{\pi}^{new} \Leftarrow \boldsymbol{\pi}^{old}\mathbf{R}\,\mathrm{diag}(\mathbf{h})$. Element-wise, this corresponds to:

  for $j = 0$ to $|\mathcal{S}| - 1$ do
    $a \Leftarrow 0$;
    for $i = 0$ to $|\mathcal{S}| - 1$ do
      $a \Leftarrow a + \boldsymbol{\pi}^{old}[i]\mathbf{R}[i, j]\mathbf{h}[j]$;
    end for;
    $\boldsymbol{\pi}^{new}[j] \Leftarrow a$;
  end for;

  • a is a high-precision accumulator

  The Jacobi method does not have guaranteed convergence, but it is usually faster than the Power method in practice.
- **(Forward) Gauss-Seidel**: $\boldsymbol{\pi}^{new} \Leftarrow \boldsymbol{\pi}^{old}\mathbf{L}(\mathrm{diag}(\mathbf{h})^{-1} - \mathbf{U})^{-1}$, where $\mathbf{L}$ and $\mathbf{U}$ are the lower and strictly upper triangular portions of $\mathbf{R}$, respectively. Element-wise, this corresponds to:

  for $j = 0$ to $|\mathcal{S}| - 1$ do
    $a \Leftarrow 0$;
    for $i = 0$ to $|\mathcal{S}| - 1$ do
      $a \Leftarrow a + \boldsymbol{\pi}^{curr}[i]\mathbf{R}[i, j]\mathbf{h}[j]$;

  • a is a high-precision accumulator

```
    end for;
    π^curr[j] ⇐ a;
  end for;
```

Note that, unlike the Power and Jacobi iterations, which require two distinct vectors, $\boldsymbol{\pi}^{old}$ and $\boldsymbol{\pi}^{new}$, Gauss-Seidel uses a single vector, $\boldsymbol{\pi}^{curr}$, since its *old* entries are updated to the *new* values one at a time, in place. The Gauss-Seidel method does not have guaranteed convergence either, but it is at least as fast as the Jacobi method, and often much faster, so it is considered the best among these three methods. Its convergence rate is affected by the order in which the states are considered.

4 Explicit Distributed Solution Approaches

Explicit distributed solution methods can make use of the workstations on a local area network to increase the amount of memory available overall, while at the same time attempting to speed up the solution process. We consider first the problem of state-space generation, basing our discussion mostly on our own work [15,35], except for the use of hashing for the mapping, which was experimented by Haverkort [28]. We should also mention the work of Caselli, Conte, and Marenzoni [14,31], one of the first groups to work along these lines. We do not consider instead works on parallel, shared-memory, algorithms, such as the one presented in [1]; these are confronted with substantially different issues. For the distributed numerical solution of linear systems, much work is available, thus we focus only on the special case of CTMC solution [32]. We should also note that preliminary attempts at using distributed solutions in conjunction with the implicit Kronecker representations of Sect. 5.2 [11,25], have also been proposed, but more work is needed in this area.

4.1 Explicit Distributed State-Space Generation

Since explicit state-space generation essentially means a breadth-first exploration of a large graph, eventually reaching each node at least once, the idea behind a distributed algorithm for state-space generation is to use multiple processes that perform this exploration concurrently on distinct nodes of the graph. Assuming we have N processors (and processes), a natural way to do this is to define a mapping

$$Proc : \mathcal{S} \rightarrow \{0, \ldots, N-1\},$$

where $Proc(\mathbf{i})$ is the *owner* of state $\mathbf{i}$, i.e., the process responsible for storing and exploring $\mathbf{i}$. This defines a partition of $\mathcal{S}$ into N sets $\mathcal{S}_{[n]} = \{\mathbf{i} : Proc(\mathbf{i}) = n\}$, for $0 \leq n < N$.

The outline of a distributed generation algorithm based on this mapping is given in Fig. 2. Each of the N processes performs a task analogous to that of the sequential algorithm, with the following differences:

- Only one of the N processes, the one with index $Proc(\mathbf{s}^{initial})$, has initially any work to do (a state to explore).

```
ExploreExplicitDistributed(n : process) : set of state
  Build and return S_[n], the portion of the state space assigned to process n.
  declare S_[n], U_[n] : set of state;
  declare i, j : state;
  declare m : process;
1. if Proc(s^initial) = n then U_[n] ⇐ {s^initial}; else U_[n] ⇐ ∅; end if;
2. S_[n] ⇐ ∅;
3. while "not received terminate message" do
4.    while ∃i ∈ U_[n] do
5.       for each j ∈ N(i) do
6.          m ⇐ Proc(j);                    • determine the process m owner of j ...
7.          if m ≠ n then
8.             SendState(m, j);             • ... if not n itself, send j to m ...
9.          elsif j ∉ U_[n] ∪ S_[n] then    • ... otherwise explore later, if new
10.            U_[n] ⇐ U_[n] ∪ {j};
11.         end if;
12.      end for;
13.      U_[n] ⇐ U_[n] \ {i};               • move i from unexplored ...
14.      S_[n] ⇐ S_[n] ∪ {i};               • ... to explored
15.   end while;
16.   U_[n] ⇐ U_[n] ∪ ReceiveStates \ S_[n]; • get states sent by other processes, if any
17. end while;
```

Fig. 2. Distributed state-space generation for process n.

- Each "destination" state **j** encountered in the innermost loop is managed locally only if it happens to be owned by the same process as the "source" state **i**; otherwise, it is sent to the correct owner.
- Occasionally, states that have been sent to a process from any of the other $N-1$ processes are retrieved and inserted in the set of unexplored states, if they have not yet been encountered before.
- Termination does not simply occur when the set of unexplored states for a given process becomes empty, since more states might be sent later to it by the other processes. Rather, a *termination detection* algorithm must determine when *all the processes have exhausted their unexplored state set and no more messages are in transit*[4].

Several aspects in the pseudo-code of Fig. 2 can be defined in more detail, such as the mechanism to send states (it might be advantageous to batch multiple states in a single message destined to a given process) and the frequency at which a process polls the receive queue for new states sent to it (we show the call to *ReceiveStates* in the outermost loop, but in practice this check might have to performed more frequently to avoid "parking" too many states in the

[4] In our studies, we used the circulating probe algorithm by Dijkstra et al. [24]; another possibility would be the scalable "non-committal barrier" described by Nicol [36].

communication buffers). However, the main decision left unspecified so far is the nature of the function used to map states to processes.

A good choice for *Proc* must be efficient to encode and compute, and should also achieve the following performance goals:

- First and foremost, the memory required to store $\mathcal{S}_{[n]}$ upon termination should be approximately the same for all n. This is fundamental, since memory is the main resource bottleneck in explicit state-space generation. A good measure for this is the *spatial balance*, which we define as the maximum among the ratios between the sizes of the sets of states allocated to any two processes (the closer this is to one, the better):

$$\max_{0 \le m,n < N} \frac{|\mathcal{S}_{[n]}|}{|\mathcal{S}_{[m]}|} \ge 1.$$

- The communication between processes should be balanced, that is, the number of states received and sent by each process should be approximately the same. Even more importantly, though, this number should be kept as small as possible, so we measure this quantity as the *fraction of cross-arcs* (the smaller the better):

$$0 \le \frac{\sum_{\mathbf{i} \in \mathcal{S}} |\{\mathbf{j} \in \mathcal{N}(\mathbf{i}) : Proc(\mathbf{i}) \ne Proc(\mathbf{j})\}|}{\sum_{\mathbf{i} \in \mathcal{S}} |\mathcal{N}(\mathbf{i})|} \le 1$$

- Finally, we would like to achieve a good *temporal balance*: most of the processes should be active most of the time, i.e., they should rarely be idle with an empty unexplored set, waiting to receive states from other processes. A good temporal balance translates into a good speedup. Of course, ignoring possible super-linear speedup effects due to virtual memory, the best we can hope to achieve is an almost linear speedup, i.e., reducing the generation time by a factor of N.

We mention three possibilities to define *Proc*.

Static User-Provided Definition. In [15] we proposed a static user-provided function based on the idea of hashing some of the state components. In particular, we assumed a Petri net model and the function was of the form

$$Proc(\mathbf{i}) = (\mathbf{i}[0] + p\,\mathbf{i}[1] + \cdots + p^{r-1}\mathbf{i}[r-1]) \bmod N$$

where p is a prime number (we used 1,013) and $\mathbf{i}[0]$ through $\mathbf{i}[r-1]$ are the number of tokens in r of the places of the Petri net for the given marking (i.e., state). We observed that the selection of the subset of places upon which we base the definition of *Proc* can affect the quality of the resulting partition. However, in all cases, using several "reasonably informed choices" for this selection, we managed to achieve good results on $N = 6$ processors: the spatial balance ranged from 1.33 to 1.38 in our experiments, while the fraction of cross-arcs ranged from 24% to 61%, and the speedup was up to 4.9 out of an ideal 6.

One advantage of letting the user specify the function is that this makes it possible to pursue specific goals. For example, we observed how, by setting p to 1 instead of a prime number and selecting r places where the tokens can only increase or decrease by one at each transition (i.e., event) firing, all cross-arcs can only exist between processes with contiguous indices, i.e., from n to $(n-1) \bmod N$ or $(n+1) \bmod N$. This property could be extremely important if the interconnection medium between the processors were a ring where any number of pairs of adjacent processors can communicate at the same time, since all communications would have to be only between adjacent processors in this case. Another important observation is that the firing in marking $\mathbf{i}$ of any transition not connected to any of the r places will lead to a marking $\mathbf{j}$ with the same owner, i.e., $Proc(\mathbf{i}) = Proc(\mathbf{j})$. This property can be exploited when attempting to minimize the number of cross-arcs, by choosing an appropriately small set of r places. Of course care must be taken in both situations: in the former case, choosing r places constituting an *invariant* of the form $\mathbf{i}[0] + \cdots + \mathbf{i}[r-1] = c$ would be disastrous, since all markings would be owned by process $c \bmod N$, while the other processes would be always idle. In the latter situation, choosing too small a set of places, or places with too small a range of possible token populations, could lead to an uneven partition, hence to a poor spatial balance.

Dynamic Automatic Re-mapping. Relying on a user-provided function requires the user to provide additional information which is related to the solution process, not to the high-level system behavior; in addition, as we just discussed, there is the risk that such a static a-priori definition results in a poor spatial balance, where most of the states are assigned to a small subsets of the processes, or a poor temporal balance, where only a few of the processes are active at any one time.

Thus, we explored a second approach [35] where the definition of *Proc* can be dynamically adjusted at run-time, to ensure that both the size of $\mathcal{U}_{[n]} \cup \mathcal{S}_{[n]}$ and that of $\mathcal{U}_{[n]}$ alone are balanced across the N processes, i.e., to ensure good spatial and temporal balance throughout the execution. This is achieved through the use of an intermediate mapping of the states to a large number C of *classes* (say, $C = 100N$), which are then mapped to processes:

$$Class : \mathcal{S} \to \{0, \ldots, C-1\} \qquad Proc : \{0, \ldots, C-1\} \to \{0, \ldots, N-1\}.$$

Even when C is of the order of thousands or tens of thousands, the mapping *Proc* can be easily stored by each process as an array of size C whose entries have value in $\{0, \ldots, N-1\}$, so only the mapping *Class* is non-trivial. In [35], instead of a user-provided function, we used the lexicographic order between states to define this mapping. The description we give here is a slight variation of this idea.

In a first phase, each of the N processes independently builds the same *control set* of $C-1$ different states found through a depth-first search using discrete-event simulation. More precisely, the N processes use a pseudo-random number generator initialized with the same seed and start exploring a path through

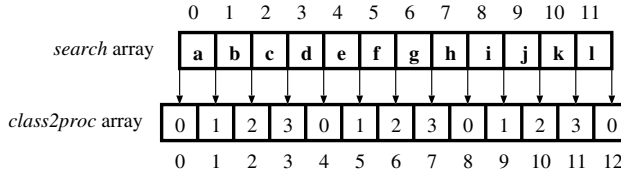


Fig. 3. The definition of the *Class* and *Proc* mappings.

the state space in such a way that, once in state $\mathbf{i}$, the next state $\mathbf{j} \in \mathcal{N}(\mathbf{i})$ to which the model transitions next is chosen with uniform probability (not according to timing specifications in the model, since this might bias the search toward the "likely" states, while, for state-space generation, all states are equally important); then, still without communicating with each other, each process sorts these states into an array "*search*" and allocates a corresponding array "*class2proc*" of size C, initialized as $\mathit{class2proc}[i] = i \bmod N$. Then, the actual state-space generation begins:

- Each process n initializes $\mathcal{U}_{[n]}$ with the states in position i of array *search*, for $\lfloor C/N \rfloor n \leq i < \lfloor C/N \rfloor (n+1)$, then begins the usual iterations.
- When a process n needs to determine the owner of a state $\mathbf{j}$, it performs a binary search for $\mathbf{j}$ on the array *search*, of which it has a copy.
- If $\mathbf{j}$ is found, this state is already known to its owner, there is nothing to do.
- Otherwise, the index $x \in \{0, \ldots, C-1\}$ where the search for $\mathbf{j}$ failed, i.e., $\mathbf{j}$'s lexicographic position with respect to the $C-1$ elements of the control set, identifies the process owner of $\mathbf{j}$: $m = \mathit{class2proc}[x]$. Thus, as usual, process n checks whether $\mathbf{j}$ is already stored in $\mathcal{U}_{[n]} \cup \mathcal{S}_{[n]}$, if $m = n$, or sends $\mathbf{j}$ to process m, if $m \neq n$.

For example, consider Fig. 3, where $N = 4$ and $C = 3N$. If the control set contains states $\{\mathbf{a}, \ldots, \mathbf{l}\}$, $\mathcal{U}_0$, $\mathcal{U}_1$, $\mathcal{U}_2$, and $\mathcal{U}_3$ will be initialized to $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$, $\{\mathbf{d}, \mathbf{e}, \mathbf{f}\}$, $\{\mathbf{g}, \mathbf{h}, \mathbf{i}\}$, and $\{\mathbf{j}, \mathbf{k}\}$, respectively. Then, during the iterations, if a process searches for a state with a lexicographic position between $\mathbf{f}$ and $\mathbf{g}$, it will determine that this state would belong in position 6 of the control set, if it were in it, but, since it is not, the process owning it is $\mathit{class2proc}[6]$, that is, 2.

We now consider the dynamic remapping aspects of this approach. The state-space generation can periodically stop and check the spatial, or temporal, balance, by comparing the size of $\mathcal{U}_{[n]} \cup \mathcal{S}_{[n]}$, or $\mathcal{U}_{[n]}$ alone, over all values of n. If an unbalance is detected, it can be corrected by dynamically rearranging the allocation of classes to processes, as long as we keep track of the contribution of each class to $|\mathcal{U}_{[n]}|$ and $|\mathcal{S}_{[n]}|$. If the *Class* mapping is fine enough, it should be possible to redefine the *Proc* mapping so that the sets $\mathcal{U}_{[n]} \cup \mathcal{S}_{[n]} = \{\mathbf{i} : \mathit{Proc}(\mathit{Class}(\mathbf{i})) = n\}$ contain approximately the same number of states, for $0 \leq n < N$. Analogously, to focus on temporal balance, we simply need to limit ourselves to the number of unexplored states owned by each process.

The reallocation procedure in [35] follows an approach where each process decides approximately how many states it wants to offload (if it is overloaded) or it is willing to receive (if it is underloaded). Then, a greedy matching algorithm is used to decide which classes to offload and the identity of their new owner; this decision is broadcast to all processes, who can then update the value of the entries in *class2proc* accordingly. Finally, any class that was reallocated from process n to process m must exchange owner, i.e., the states in it must be actually transferred.

The tradeoff between frequent checks and a more sensitive triggering condition for a reallocation decision versus less frequent checks and being willing to accept a larger unbalance is clear: the former is more likely to ensure even memory requirements and good processor utilization, but at the cost of larger and more frequent overhead phases during which no useful exploration takes place.

In our experiments [35], we found that dynamic re-balancing was quite beneficial when using 16 processors, while the improvement was minor when using eight processors. We also observed that the overhead as a percentage of the overall runtime was somewhat higher when the goal was temporal balance rather than spatial balance, although it was in any case below 5% except when using very frequent checks. On the other hand, the best speedup on 16 processors was achieved when balancing the temporal load, almost 13 using five re-mapping phases throughout the state-space generation, while the speedup when balancing the spatial load was less than 12.

This dynamic reallocation approach is quite resilient, as long as the classes of equivalence defined by *Class* are fine enough: this is the reason for requiring a value $C \gg N$. While we did not experience a problem in our experiments, one or more of these classes could still be too large. In this case, the appropriate step would be to break down these large classes further, splitting them into multiple classes; of course, this requires enlarging the control set, hence extending the *search* and *class2proc* arrays.

Hashing for State Storage and Mapping. In a sequential approach, a hash table is a reasonable alternative to a search tree for storing states and being able to determine whether a state is new or already in $\mathcal{U} \cup \mathcal{S}$. The main problem in using a hash table for this application is that the size of $\mathcal{S}$ cannot be predicted in advance, but this can be remedied by resizing the hash table when the number of collisions grows too much.

The same is true for a distributed approach as well, where we can use a hash table for each of the N processes, and still require a separate state-to-process mapping. Alternatively, since we already need a hashing function to store a state, we can simply rely on this function to determine the process as well. Conceptually, this requires us to use a single hash table of size HN, split into N equal portions that physically reside in the corresponding N processes. Then, the hash function

$$Hash : \mathcal{S} \rightarrow \{0, \ldots, HN - 1\}$$

can be used by process n to determine the owner m of a state $\mathbf{j}$: $m = \lfloor Hash(\mathbf{j})/H \rfloor$. If $n = m$, process n can use the same function to determine the position $Hash(\mathbf{j})$ mod H where $\mathbf{j}$ should be placed in its portion of the hash table.

As one would expect, a completely hash-based approach achieves a reasonably good spatial balance (the authors of [28] report a value of 1.49 for their experiment), but the number of cross-arcs is harder to control (about 50% in [28], using the idea of restricting the definition of the hashing function to a subset of the places of their Petri net model).

4.2 Explicit Distributed Markov Chain Generation and Solution

The distributed generation of the entries of matrix $\mathbf{R}$ follows the same principles as for the sequential approach, with two main differences. First, the mapping Ψ assigning indices to the states is more complex because it must indicate both the identity of the state and of the process owning it. A reasonable approach is then to define N per-process mappings

$$\Psi_{[n]} : \mathcal{S}_{[n]} \rightarrow \{0, \ldots, |\mathcal{S}_{[n]}| - 1\}$$

so that the overall mapping Ψ is given by

$$\Psi(\mathbf{i}) = \left(Proc(\mathbf{i}), \Psi_{[Proc(\mathbf{i})]}(\mathbf{i})\right).$$

A second, related, difference is the management and storage of the entries, since, if the value $\Psi_{[n]}(\mathbf{i})$ is computed using either discovery or lexicographic order (as it is normally the case), only process n can compute its value.

Storing the Transition Rate Matrix: By Rows or by Columns? If we want to generate and store the entries of $\mathbf{R}$ by rows, i.e., process n stores all entries $\mathbf{R}[(n,i),(m,j)]$, these are the actions process n must perform when it is exploring state $\mathbf{i}$ with index $i = \Psi_{[n]}(\mathbf{i})$:

for each transition from $\mathbf{i}$ to $\mathbf{j}$ with rate λ
  process n computes $m = Proc(\mathbf{j})$
  if $m = n$ then
    process n computes $j = \Psi_{[n]}(\mathbf{j})$
  else
    process n sends the pair $\langle n, \mathbf{j} \rangle$ to process m
    process m computes $j = \Psi_{[m]}(\mathbf{j})$
    process m returns the pair $\langle m, j \rangle$ to process n
  endif
  process n sets $\mathbf{R}[(n,i),(m,j)]$ to λ

If instead we want to store the entries by columns, i.e., process m stores all entries $\mathbf{R}[(n,i),(m,j)]$, process n must perform the following actions when exploring state $\mathbf{i}$ with index $i = \Psi_{[n]}(\mathbf{i})$:

```
for each transition from i to j with rate λ
  process n computes m = Proc(j)
  if m = n then
    process n computes j = Ψ_[n](j)
    process n sets R[(n,i),(m,j)] to λ
  else
    process n sends the tuple ⟨n,i,j,λ⟩ to process m
    process m computes j = Ψ_[m](j)
    process m sets R[(n,i),(m,j)] to λ
  endif
```

(in these statements, it might be more correct to say "increments $\mathbf{R}[(n,i),(m,j)]$ by λ" instead of "sets $\mathbf{R}[(n,i),(m,j)]$ to λ", since multiple ways to transition from $\mathbf{i}$ to $\mathbf{j}$ might exist). Thus, storing $\mathbf{R}$ by columns is not only preferable in the subsequent numerical solution algorithms, but it also cuts in half the number of logical messages that need to be sent when generating the entries of $\mathbf{R}$, since process n does not require an answer from process m. We observe that, in [15], we generated both $\mathcal{S}$ and $\mathbf{R}$ in a single step. This is possible when the mapping $\Psi_{[n]}$ is based upon the discovery order (more precisely, the order in which states in $\mathcal{S}_{[n]}$ become known to process n), but it requires the use of more dynamic data structures, such as linked lists, to store the columns of $\mathbf{R}$, since the number of nonzero entries process n might have to store is not known beforehand. Alternatively, we can follow the same two-phase approach discussed for the sequential case, where the N processes just count the number of entries while generating the state space, in the first phase, and fill these entries in the second phase.

Block Methods for the Numerical Solution. For notational simplicity, we define the sub-matrices

$$\mathbf{R}_{[n,m]} =_{\text{df}} \mathbf{R}[\Psi(\mathcal{S}_{[n]}), \Psi(\mathcal{S}_{[m]})].$$

Assuming storage by columns, the entries of $\mathbf{R}$ are distributed so that process m stores $\mathbf{R}_{[n,m]}$, for $0 \leq n < N$. At this point, the distributed numerical solution can begin. Since, as we stressed already, memory is the bottleneck, process n should store only the portions $\mathbf{h}_{[n]} =_{\text{df}} \mathbf{h}[\Psi(\mathcal{S}_{[n]})]$ and $\boldsymbol{\pi}_{[n]} =_{\text{df}} \boldsymbol{\pi}[\Psi(\mathcal{S}_{[n]})]$ of $\mathbf{h}$ and $\boldsymbol{\pi}$, and the same should hold for any other auxiliary vectors required by the solution method, to allow the size of the models that can be solved to scale linearly in N.

In such a setting, it is natural to employ so-called *block methods* for the numerical solution. For example, the computation performed by process n in a block-Jacobi iteration is

$$\boldsymbol{\pi}_{[n]}^{new} = \left(\sum_{0 \leq m < N} \boldsymbol{\pi}_{[m]}^{old} \mathbf{R}_{[m,n]} \right) \text{diag}(\mathbf{h}_{[n]}). \tag{2}$$

$\boldsymbol{\pi}_{[n]}^{old} \Leftarrow$ "*initial guess*";
while "*keep doing global synchronizations*" do
  $BroadcastVector(\boldsymbol{\pi}_{[n]}^{old})$;
  $\mathbf{a} \Leftarrow \mathbf{0}$; • $\mathbf{a}$ is a local auxiliary vector of size $|\mathcal{S}_{[n]}|$
  for each $m \neq n$ do
    $\boldsymbol{\pi}_{[m]}^{old} \Leftarrow ReceiveBroadcastVectorFrom(m)$;
    $\mathbf{a} \Leftarrow \mathbf{a} + \boldsymbol{\pi}_{[m]}^{old}\mathbf{R}_{[m,n]}$;
  end for;
  while "*keep doing local iterations*" do
    $\boldsymbol{\pi}_{[n]}^{new} \Leftarrow \left(\boldsymbol{\pi}_{[n]}^{old}\mathbf{R}_{[n,n]} + \mathbf{a}\right) \mathrm{diag}(\mathbf{h}_{[n]})$;
    $\boldsymbol{\pi}_{[n]}^{old} \Leftarrow \boldsymbol{\pi}_{[n]}^{new}$;
  end while;
end while;

Fig. 4. The iterations performed by process n in a distributed block-Jacobi scheme.

However, process n does not store $\boldsymbol{\pi}_{[m]}^{old}$ for $m \neq n$, so this information must be exchanged between processes, and this can be quite time consuming. If one-to-many communication is possible, N broadcasts are required, where process n sends $\boldsymbol{\pi}_{[n]}^{old}$ to all other processes; otherwise, $N(N-1)$ one-to-one communications are required, for all pairs (m, n) with $m \neq n$.

To reduce the cost of communication as a fraction of the total solution time, the iterative scheme shown in Fig. 4 can be used. In the outer while-loop, each of the N processes broadcasts its current portion $\boldsymbol{\pi}_{[n]}^{old}$ of the probability vector and computes the contributions of all the other portions $\boldsymbol{\pi}_{[m]}^{old}$, for $m \neq n$, to the right-hand-side value in Eq. 2. Then, in the inner while-loop, process n uses a traditional iteration (we show a variant based on the Jacobi method, but one based on the Gauss-Seidel method would be possible as well), where only the entries of $\boldsymbol{\pi}_{[n]}^{old}$ are updated, while the ones of $\boldsymbol{\pi}_{[m]}^{old}$ are effectively kept constant. Thus, there is a tradeoff between more frequent global synchronizations, which imply more communication overhead, vs. less frequent ones, which imply using older data in the local iterations and potentially slowing down the numerical convergence.

We do not discuss the issue of distributed solution further. Our reason for introducing it was simply to illustrate that it is possible to store only vectors of size $|\mathcal{S}_{[n]}|$ (in the method we presented, four are needed: the current and new iterate, the portion of the holding time vector $\mathbf{h}_{[n]}$, and an auxiliary vector $\mathbf{a}$), in addition to the blocks of $\mathbf{R}$ corresponding to columns in $\mathcal{S}_{[n]}$ and to the portions of the probability vector received from other processes (these, however, can be "used and discarded on the fly" while computing $\mathbf{a}$, so, in principle, impose no additional storage requirements). Distributed solution methods, with either synchronous communication (such as the one we illustrated, where all processes exchange data at the same time) or asynchronous communication, and

their convergence properties are an active area of research and a more thorough investigation of the state of the art is beyond the scope of this presentation (see for example [32] and references within).

5 Implicit Sequential Solution Approaches

We now turn to *implicit* methods for the storage of the state space and the transition rate matrix. By this we mean methods that exploit certain symmetries, present to some extent in the state space of any system exhibiting some asynchronous behavior, and use data structures that normally require much less than linear space. Another way to state this is that, with explicit methods, we can point at a specific memory location corresponding to a given state in $\mathcal{S}$ or a given entry of $\mathbf{R}$, while, with implicit methods, a state or an entry must be "reconstructed" using information present in several memory locations. As we will see, this results in huge memory savings, and might also result in time savings (as it is normally the case for state-space generation), or it might instead imply an overhead (as is often the case for Markov chain solution).

In all cases, we assume that the model is composed of (or decomposed into) K sub-models. More precisely, this means that $\widehat{\mathcal{S}}$ is the cross-product of K *local state spaces*: $\widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$. The assumption of such a structure in the model is quite reasonable: indeed it is almost always the case that systems, especially complex ones, are built of interacting components. We indicate with n_k the cardinality of $\mathcal{S}_k$, for $K \geq k \geq 1$, and stress that $\mathcal{S}$, hence n_k, might be known before exploring $\mathcal{S}$, or might become known only afterwards (the latter assumption complicates matters only slightly). From now on, we assume that the local states of $\mathcal{S}_k$ are stored (once) and *indexed* separately, so that we identify a (global) state with the K-tuple of its K local states indices. Thus, a state can be stored in $\sum_{K \geq k \geq 1} \lceil \log n_k \rceil$ bits. We also identify a state with its *mixed-base value*:

$$\mathbf{i} \equiv (\mathbf{i}_K, \ldots, \mathbf{i}_1) = (\ldots((\mathbf{i}_K) \cdot n_{K-1} + \mathbf{i}_{K-1}) \cdot n_{K-2} \cdots) \cdot n_1 + \mathbf{i}_1 = \sum_{K \geq k \geq 1} \mathbf{i}_k \cdot n_{k-1:1},$$

where $n_{k:l} =_{\text{df}} n_k \cdot n_{k-1} \cdots n_l$, for $k \geq l$, and $n_{k:l} =_{\text{df}} 1$, for $k < l$. The reason for counting the sub-models from K down to 1 should now be clear: the K^{th} index is the "most significant digit". For example, if $K = 3$, $|\mathcal{S}_3| = 4$, $|\mathcal{S}_2| = 3$, and $|\mathcal{S}_1| = 5$, the mixed-base value of global state $(2, 0, 3)$ is $2 \cdot n_{2:1} + 0 \cdot n_{1:1} + 3 \cdot n_{0:1} = 2 \cdot (3 \cdot 5) + 0 \cdot (5) + 3 = 33$.

5.1 Implicit Sequential State-Space Generation

In the area of formal methods, and in particular of model checking [22], *binary decision diagrams* (*BDDs* [5,6]), have been successfully used to generate and store enormous state spaces [13]. We are going to employ a non-binary version of decision diagrams whose definition [29] can be seen as an extension from the binary to the general discrete world, but can also be reached starting from the

explicit data structure we introduced in [19]. We take this second point of view [16,17,33].

An Explicit Multi-level Data Structure. The *multi-level* explicit data structure introduced in [19] can be used in our procedure *ExploreExplicitSequential* to store $\mathcal{S}$ as it is being built. The idea is that, since a state $\mathbf{i} \in \mathcal{S}$ is identified with a K-tuple $(\mathbf{i}_K, \ldots, \mathbf{i}_1)$ of local state indices, we can use K levels of search trees as shown in Fig. 5, where we assume $K = 4$. To search for a state $\mathbf{i} \equiv (\mathbf{i}_K, \ldots, \mathbf{i}_1)$ in the current $\mathcal{S}$, we first search for $\mathbf{i}_K$ in the only search tree at the top level, K. If $\mathbf{i}_K$ is found, we follow the corresponding pointer to a tree at level $K-1$, and search $\mathbf{i}_{K-1}$ in this tree, and so on. The process terminates either if we find $\mathbf{i}_1$ in the tree at level 1 reached according to the path determined by $(\mathbf{i}_K, \ldots, \mathbf{i}_2)$, in which case we conclude that $\mathbf{i}$ is already in $\mathcal{S}$, or if we fail to find $\mathbf{i}_k$ in the corresponding tree at level k, for some k, $K \geq k \geq 1$, in which case we conclude that $\mathbf{i}$ is not yet in $\mathcal{S}$, and we know the point at which to start inserting the missing portion $(\mathbf{i}_k, \ldots, \mathbf{i}_1)$ of the state.

In [19], we pointed out two advantages of this data structure with respect to others used in explicit approaches. First, the storage required for it is mostly due to the bottom level, as long as there is a sufficient fan-out from level to level, i.e., as long as each tree at each level has at least several nodes. The overall number of nodes at the bottom level is exactly the number of states, $|\mathcal{S}|$, and to store them we can use, in principle, only integers and pointers of size $\lceil \log n_1 \rceil$ bits. Assuming a binary tree, this means that, for most practical models, we can store $\mathcal{S}$ in little over $3|\mathcal{S}|\lceil \log n_1 \rceil$ bits, as opposed to $3|\mathcal{S}| \sum_{K \geq k \geq 1} \lceil \log n_k \rceil$ bits. A second property of this data structure results instead in an improved execution time. Assume that we know the pointers $T_K, \ldots, T_1$ to the trees where we found $\mathbf{i}_K, \ldots, \mathbf{i}_1$, respectively, and that $\mathbf{j} \in \mathcal{N}(\mathbf{i})$. To determine whether $\mathbf{j}$ is in the currently-known portion of the state space, *ExploreExplicitSequential* needs to determine whether there is a path labelled with $(\mathbf{j}_K, \ldots, \mathbf{j}_1)$ in the multi-level data structure. However, the nature of the model might imply that $\mathbf{j}$ can differ from $\mathbf{i}$ only in components with index k or lower (for example, this happens if $\mathbf{j}$ is reached from $\mathbf{i}$ through the firing of an event e that only affects some of the sub-models, a very common situation). When this is the case, the first $K-k$ components of the two states coincide, that is, $(\mathbf{i}_K, \ldots, \mathbf{i}_{k+1}) = (\mathbf{j}_K, \ldots, \mathbf{j}_{k+1})$, thus the search can start at the tree pointed by T_k, instead of the top tree. Experimentally, we showed that, on a range of common applications, exploiting this *locality* property can result in considerable execution time reductions, at no extra cost in memory requirements.

Multi-valued Decision Diagrams. The *multi-valued decision diagram* (MDD) definition we adopt is conceptually obtained from the explicit multi-level data structure by recognizing common subtrees at the same level and merging them, in a bottom-up fashion. More formally, a (*quasi-reduced ordered*) MDD [17] is a directed acyclic multi-graph where:

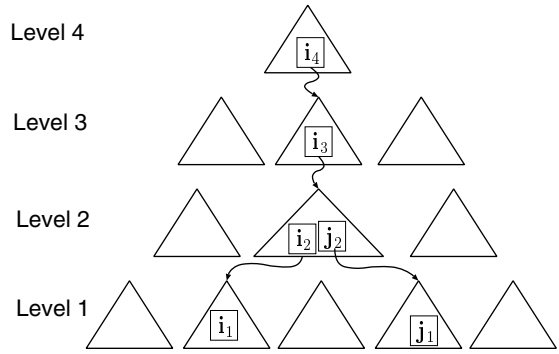


Fig. 5. The multi-level explicit data structure introduced in [19] ($K = 4$).

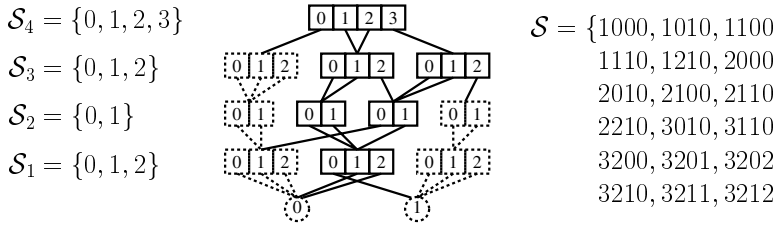


Fig. 6. An example MDD and the state space $\mathcal{S}$ encoded by it.

- Nodes are organized into $K + 1$ *levels*. We write $\langle k.p \rangle$ to denote a generic node, where k is the level and p is a unique index for the nodes at that level. Level K contains only a single *non-terminal* node $\langle K.r \rangle$, the *root*, whereas levels $K-1$ through 1 contain one or more non-terminal nodes. Level 0 consists of two *terminal* nodes, $\langle 0.0 \rangle$ and $\langle 0.1 \rangle$.
- A non-terminal node $\langle k.p \rangle$ has n_k arcs pointing to nodes at level $k-1$. If the i^{th} arc, for $i \in \mathcal{S}_k$, is to node $\langle k-1.q \rangle$, we write $\langle k.p \rangle[i] = q$.
- A non-terminal node cannot *duplicate* (i.e., have the same pattern of arcs as) another node at the same level.

Clearly, merging common trees transforms the tree into a directed acyclic graph, but preserves the logic of state search: we can still start from the top level and determine whether a given state belongs to $\mathcal{S}$ or not, by following the corresponding path and determining whether it leads to $\langle 0.1 \rangle$ or $\langle 0.0 \rangle$. For example, Fig. 6 shows a four-level MDD and the set $\mathcal{S}$ encoded by it (paths from the root to the node $\langle 0.1 \rangle$ describe the reachable states). The figure shows arrays being used, instead of search trees: this is more efficient provided a good portion of the array entries is actually used for paths corresponding to reachable states, and it is possible provided that we know a priori the size of each $\mathcal{S}_k$, or are willing to use *dynamic arrays*. Paths leading to $\langle 0.0 \rangle$, which of course

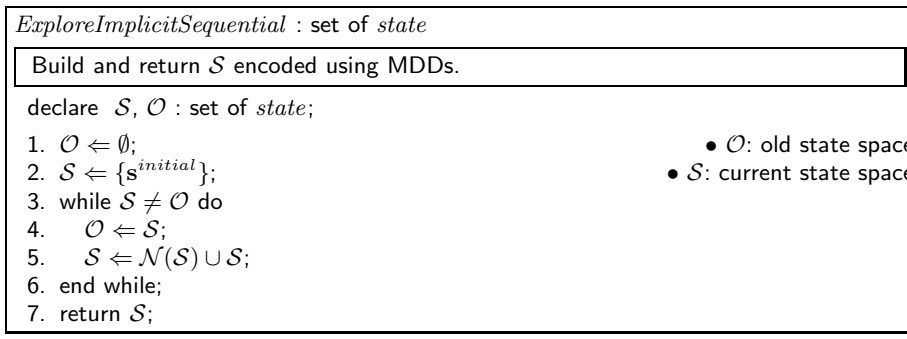


Fig. 7. An implicit sequential procedure to generate $\mathcal{S}$.

correspond to unreachable states, are present in Fig. 6 because we use arrays, while in the analogous explicit multi-level data structure of Fig. 5 we simply have incomplete paths, because we use search trees. In fact, we showed how, in an actual implementation, it is possible to avoid storing nodes at any level k corresponding to a logical zero or one, i.e., nodes encoding $\emptyset$ or $\mathcal{S}_k \times \cdots \times \mathcal{S}_1$, respectively, by reserving the node identifiers $\langle k.0 \rangle$ and $\langle k.1 \rangle$ for this purpose (in Fig. 6, these nodes and the arcs emanating from them are shown with dotted lines).

The enormous success of decision diagrams (in the literature, mostly *binary* decision diagrams, or BDDs, have been considered) is not just due to ability of *storing* state spaces in a more compact way, but also of *generating* them more efficiently. This is because we do not use an explicit exploration approach that adds just one state at a time to $\mathcal{S}$. Rather, using implicit techniques, we add entire subsets of states in a single operation, as shown in Fig. 7. Considering statement 5, it is clear that we must be able to efficiently compute both the set of states reachable in one step from the currently-know state space, $\mathcal{N}(\mathcal{S})$, and their union $\mathcal{N}(\mathcal{S}) \cup \mathcal{S}$. Efficient algorithms are known to compute the union of two sets encoded as MDDs, so the focus of much research has been on the encoding and computation of the next-state function.

Encoding the Next-State Function. One popular way to encode the next-state function is to use a decision diagram for it as well, where both the current and the next state are found on a path from the root to $\langle 0.1 \rangle$ (thus the diagram has $2K$ levels, usually interleaved for greater efficiency). Rather than following this traditional approach, however, we introduce a different one [16,17,33] that is both much more efficient, and also strongly related to the implicit encoding of the transition rate matrix, which we will consider in Sect. 5.2.

The idea is based on the fact that many systems exhibit what can be called a *globally-asynchronous locally-synchronous behavior*, that is, most events are enabled by and affect only a small subset of the K sub-models, which must then be "synchronized", while distinct events can occur concurrently and asynchronously in different parts of the system. We then require that, for each event

e, its next-state function $\mathcal{N}_e$ can be written as the cross-product of K local functions:

$$\mathcal{N}_e = \mathcal{N}_{K,e} \times \cdots \times \mathcal{N}_{1,e}, \tag{3}$$

where $\mathcal{N}_{k,e} : \mathcal{S}_k \to 2^{\mathcal{S}_k}$. This *product-form* requirement is quite natural for two reasons. First, many modeling formalisms satisfy it (for example, we showed in [21] that any Petri net model conforms to this behavior for any partition of its places). Second, if a given model does not respect the product-form behavior, we can always coarsen the sub-models or refine $\mathcal{E}$ so that it does (in the limit, this results in having a single model, i.e., $K = 1$, or defining a different event for each state-to-state transition, i.e., $\mathcal{E} = \mathcal{A}$, but this does not seem to occur in practice).

When an event e is independent of a sub-model (or level) k, $\mathcal{N}_{k,e}$ is the identity, that is:

$$\forall \mathbf{i}_k \in \mathcal{S}_k, \quad \mathcal{N}_{k,e}(\mathbf{i}_k) = \{\mathbf{i}_k\}. \tag{4}$$

When this is not the case, we say instead that e *depends* on level k. We let $First(e)$ and $Last(e)$ be the first and last levels on which event e depends and, as we shall see, one of the goals of a good decomposition is to have events span few levels, that is, to define sub-models so that the range $First(e) - Last(e) + 1$ is small with respect to K. In particular, events e such that $First(e) = Last(e) = k$ are said to be *local*, while those where $First(e) > Last(e)$ are said to be *synchronizing*. Note that the local events at a given level can be merged into a single *macro-event* λ_k without violating the product-form requirement, since we can define

$$\mathcal{N}_{\lambda_k} = \mathcal{N}_{K,\lambda_k} \times \cdots \times \mathcal{N}_{1,\lambda_k}$$

where $\mathcal{N}_{k,\lambda_k} = \bigcup_{e:First(e)=Last(e)=k} \mathcal{N}_{k,e}$, where, again, the union is applied to the value of the functions, i.e., $\mathcal{N}_{k,\lambda_k}(\mathbf{i}_k) = \bigcup_{e:First(e)=Last(e)=k} \mathcal{N}_{k,e}(\mathbf{i}_k)$, while $\mathcal{N}_{l,\lambda_k}(\mathbf{i}_l) = \{\mathbf{i}_l\}$ for $l \neq k$ and $\mathbf{i}_l \in \mathcal{S}_l$. From now on, we assume that the set of events $\mathcal{E}$ has been redefined to reflect this merging.

Under these conditions, the next-state function $\mathcal{N}$ is fully captured by $(L + 1)K$ boolean matrices $\mathbf{B}_{k,e}$, where L is the number of synchronizing events; indeed, in a good decomposition, a majority of these matrices are the identity, so they do not need to be stored explicitly. Also, for any matrix matrix $\mathbf{B}_{k,e}$ that must be actually stored, we can use a sparse row-wise data structure. Then, given $\mathbf{i}_k$, we can retrieve the set $\mathcal{N}_{k,e}(\mathbf{i}_k)$ in time and memory proportional to $|\mathcal{N}_{k,e}(\mathbf{i}_k)|$, which, in practical applications, is much smaller than n_k. In other words, the storage required to encode $\mathcal{N}$ is a negligible portion of the overall storage requirements, and the time required to build its encoding is also very small.

Generating $\mathcal{S}$ as the Fixed-Point of $\mathcal{N}$. Using our encoding of the next-state function, we are able to generate enormous state spaces efficiently using a

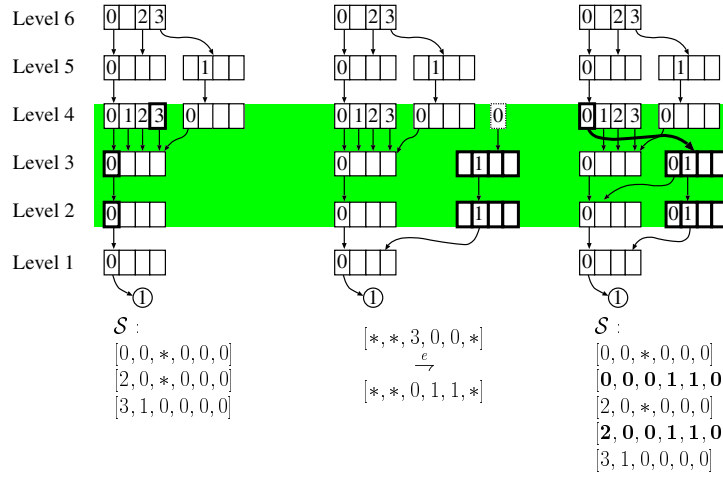


Fig. 8. Firing an event e dependent only on levels 4, 3, and 2.

modification of the algorithm shown in Fig. 7. Specifically, instead of computing $\mathcal{N}(\mathcal{S})$ at each iteration, we compute several "lighter" next-state functions, one per event, exploiting the concept of *event locality* to limit our computation to nodes at the appropriate levels. In other words, we don't even need to explore the firing an event e at level k, if $k > First(e)$ or $Last(e) > k$. Fig. 8 illustrates, as an example, the firing of an event e dependent on levels 4, 3, and 2 only, starting from the state space shown on the left (nodes encoding $\emptyset$ and arcs pointing to them are omitted for clarity). Event e is enabled only when the the sub-models 4, 3, and 2 are in the local states with index 3, 0, and 0, and its firing changes these local states to 0, 1, and 1, respectively. The state space after this firing is shown on the right. Note that a single firing adds the two states shown in boldface.

Another related improvement is the use of an efficient *iteration strategy*, where we explore the firing of events in a specific order: starting from an initial set of states (in our case, $\{\mathbf{s}^{initial}\}$), we fire exhaustively any event e for which $First(e) = 1$ (of course, only the local macro-event λ_1 satisfies this requirement), then we fire exhaustively any event e for which $First(e) = 2$ (this includes both the local macro-event λ_2 and any event synchronizing level 1 and level 2) and, for any new node (hence for the corresponding states) that these firings might create at level 1, we again fire exhaustively any event e for which $First(e) = 1$, and so on. Once we reach level K and exhaustively fire any event at level K, plus any event at lower levels on any newly created node, the process is completed: our MDD is *saturated* and it encodes exactly $\mathcal{S}$.

We introduced this idea of saturating the nodes of the MDD during fixed-point exploration in [17], and showed how, paired with locality, it can reduce the memory and time requirements to generate the state space of practical models by orders of magnitude. For example, with our implementation in SMART [18], we

could generate the state space corresponding to the famous dining philosophers problem, with 1,000 philosophers ($\mathcal{S}$ contains almost 10^{627} states!), in less than one second on an 800 MHz Pentium workstation. This represents an enormous improvement with respect to more conventional symbolic methods based on a BDD encoding of both the state space and the next state function [37,38,39,43]. Furthermore, the benefits increase with the height of the MDDs (the number of levels K), since, when decomposing practical models, the range of levels on which an event depends is going to be mostly a small constant, regardless of the value of K.

5.2 Implicit Sequential Markov Chain Generation and Solution

Armed with our understanding of the structure imposed on the state space by the logic product-form behavior, we can now discuss an analogous concept for the description of the transition rate matrix of the CTMC underlying a high-level model decomposed into sub-models. It is interesting to note, however, that work on this *Kronecker-based description* predates that on MDDs; in fact, it was our inspiration for it, rather than the other way around.

The use of Kronecker (also called *tensor*) algebra [4,23,27] for the description of a transition rate matrix is over twenty years old [2], but its real impact was realized when this approach began being applied to high-level models, first by Plateau and Stewart on *Synchronized Automata Networks* [26,41,42], then by Donatelli on the more general *Superposed Stochastic Automata* [25] and by Buchholz and Kemper [7,8,9,12] on several classes of hierarchical formalisms, including queueing networks and their variants. For a brief description of the Kronecker product "$\otimes$" and Kronecker sum "$\oplus$" operators, see the Appendix.

Kronecker Description of the Transition Rate Matrix. We have seen that the transition rate from state $\mathbf{i}$ to state $\mathbf{j}$ can be expressed in a per-event fashion:

$$Rate(\mathbf{i},\mathbf{j}) = \sum_{e\in\mathcal{E}} Rate_e(\mathbf{i},\mathbf{j}).$$

To apply our implicit description techniques, we only need to assume a product-form requirement analogous to that of Eq. 3:

$$Rate_e = Rate_{K,e} \cdots Rate_{1,e},$$

where each non-negative function $Rate_{k,e} : \mathcal{S}_k \times \mathcal{S}_k \to \mathbb{R}$ expresses the (multiplicative) contribution of sub-model k to the rate of event e (of course, dimensionally, only their product is a rate).

As it should be expected, we require that $Rate_{k,e}(\mathbf{i}_k,\mathbf{j}_k) = 0$ iff $\mathbf{j}_k \notin \mathcal{N}_{k,e}(\mathbf{i}_k)$ and we say that event e is (stochastically) independent of level k if $Rate_{k,e}$ is the identity, i.e., $Rate_{k,e}(\mathbf{i}_k,\mathbf{j}_k) = 0$ when $\mathbf{i}_k \neq \mathbf{j}_k$, 1 when $\mathbf{i}_k = \mathbf{j}_k$. Note that the logic independence of Eq. 4 is a necessary but not sufficient condition for this new definition of independence, since the local state of a sub-model k could

affect the timing of an event e but not its enabling or its effect, i.e., we might have $\mathcal{N}_{k,e}(\mathbf{i}_k) = \{\mathbf{i}_k\}$ for all $vi_k \in \mathcal{S}_k$, but the (positive) value of $Rate_{k,e}(\mathbf{i}_k, \mathbf{i}_k)$ could nevertheless depend on the particular local state $\mathbf{i}_k$. Also, we say that e is a local event for level k iff k is the only level for which $Rate_{k,e}$ is not the identity, and again we merge all such events into a macro event λ_k.

We can then define the matrices $\mathbf{W}_{k,e}$ as the encoding of $Rate_{k,e}$, for any synchronizing event e, which we assume indexed from 1 to L, and the matrices $\mathbf{R}_k$ as the encoding of $Rate_{k,\lambda_k}$:

$$\mathbf{W}_{k,e}[\mathbf{i}_k, \mathbf{j}_k] = Rate_{k,e}(\mathbf{i}_k, \mathbf{j}_k) \qquad \mathbf{R}_k[\mathbf{i}_k, \mathbf{j}_k] = Rate_{k,\lambda_k}(\mathbf{i}_k, \mathbf{j}_k),$$

and write

$$\mathbf{R} = \widehat{\mathbf{R}}[\mathcal{S}, \mathcal{S}] = \sum_{e \in \mathcal{E}} \widehat{\mathbf{R}}_e[\mathcal{S}, \mathcal{S}] = \left(\bigoplus_{K \geq k \geq 1} \mathbf{R}_k + \sum_{e=1}^{L} \bigotimes_{K \geq k \geq 1} \mathbf{W}_{k,e} \right) [\mathcal{S}, \mathcal{S}], \qquad (5)$$

where we recall that, when inexing the "potential" marix $\widehat{\mathbf{R}}$, a global state $\mathbf{i}$ is interpreted as its mixed-base integer value when used as a matrix or vector index. An analogous expression exists for $\mathbf{Q}$, the infinitesimal generator.

In other words, we can express the (huge) transition rate matrix $\mathbf{R}$ as the sub-matrix corresponding to the reachable states $\mathcal{S}$ of a (possibly even larger) matrix that can be expressed through Kronecker operators applied to $(L+1)K$ small real matrices: a huge memory saving. Furthermore, as in the logic case, many of these matrices will be the identity in practical applications. However, unlike the results we discussed for decision diagrams, iterative numerical methods multiply the implicitly-encoded matrix $\mathbf{R}$ by "explicit" probability vectors, which are now the main memory bottleneck, so we must examine how this encoding affects the run-time efficiency.

Potential vs. Actual State Space, Row vs. Column Access. One potential source of overhead when using Eq. 5 to encode $\mathbf{R}$ is that we need to consider a sub-matrix of a Kronecker expression. Initial proposals [25,41] used algorithms that can (almost) ignore the difference between the potential state space $\widehat{\mathcal{S}}$ and the actual state space $\mathcal{S}$. This is possible if we access $\mathbf{R}$, or more precisely $\widehat{\mathbf{R}}$, by rows, since, by definition, an unreachable state $\mathbf{j}$ cannot be reached from a reachable state $\mathbf{i}$, that is, $\widehat{\mathbf{R}}[\mathcal{S}, \widehat{\mathcal{S}} \setminus \mathcal{S}] = \mathbf{0}$. Unfortunately, the converse is not true, that is, $\widehat{\mathbf{R}}[\widehat{\mathcal{S}} \setminus \mathcal{S}, \mathcal{S}]$ is not necessarily zero. The Jacobi iteration can be rewritten to enforce access by rows:

```
for each i ∈ Ŝ such that π̂^old[i] > 0 do
  for each j ∈ Ŝ such that R̂[i,j] > 0 do          • access row i of R̂
    π̂^new[j] ⇐ π̂^new[j] + π̂^old[i] R̂[i,j] ĥ[j];
  end for;
end for;
```

where $\widehat{\mathbf{h}}$, $\widehat{\boldsymbol{\pi}}^{old}$, and $\widehat{\boldsymbol{\pi}}^{new}$ are vectors of size $|\widehat{\mathcal{S}}|$, indexed by the mapping

$$\widehat{\Psi} : \widehat{\mathcal{S}} \to \{0, \ldots, |\widehat{\mathcal{S}}| - 1\},$$

which is simply the mixed-based value of the state. This Jacobi iteration moves the probability mass only to states reachable from those having an initial probability mass so, if $\widehat{\boldsymbol{\pi}}^{old}$ is properly initialized according to the initial state, only entries of $\widehat{\boldsymbol{\pi}}^{old}$ corresponding to states in $\mathcal{S}$ will ever become positive, hence only entries in $\widehat{\mathbf{R}}[\mathcal{S}, \mathcal{S}]$ are used.

The disadvantages of such a "potential" approach, however, are numerous: $\widehat{\mathbf{h}}$, $\widehat{\boldsymbol{\pi}}^{old}$, and $\widehat{\boldsymbol{\pi}}^{new}$ may require much more storage than their "actual" siblings $\mathbf{h}$, $\boldsymbol{\pi}^{old}$, and $\boldsymbol{\pi}^{new}$; to make things worse, the entire vector $\widehat{\boldsymbol{\pi}}^{new}$ is used as an accumulator, so it should in principle be allocated using a higher-precision floating point type, while, with access by columns, only one scalar accumulator is required; finally, the approach just described can be used for the Jacobi method (or the even slower Power method), but the preferred Gauss-Seidel essentially requires access by columns.

Another potential overhead is the actual computation of the entry $\widehat{\mathbf{R}}[\mathbf{i}, \mathbf{j}]$ using the Kronecker expression of Eq. 5. With vectors of size $|\widehat{\mathcal{S}}|$, the indexing function $\widehat{\Psi}$ is indeed easy to define and compute, but it can still involve an overhead factor $O(K)$; an analogous overhead is potentially implied also by the multiplications of the K entries in the appropriate $\mathbf{W}_{k,e}$ matrices (in most cases, only one event causes an entry of $\widehat{\mathbf{R}}$ to be positive). Algorithms that attempt to amortize these computations have been introduced [10]; the best ones in practice appear to be based on an interleaving of the row and column indices of the K matrices, which imply an access pattern that is neither by rows nor by columns: again, they allow us to employ methods such as Power and Jacobi, but they precludes us from using Gauss-Seidel. It should also be mentioned that, in the case where the matrices $\mathbf{W}_{k,e}$ are quite dense, the *shuffle* algorithm is extremely efficient: for full matrices, the complexity of computing $\mathbf{x} \cdot \bigotimes_{K \geq k \geq 1} \mathbf{A}_k$ is $n_{K:1} \cdot \sum_{K \geq k \geq 1} n_k$ instead of $(n_{K:1})^2$ [26]; however, the matrices $\mathbf{W}_{k,e}$ are usually *ultra-sparse*, i.e., they often have around one nonzero entry per row on average, so these savings might not be realized in practice [10].

More recent implementations based on the actual state space that access $\widehat{\mathbf{R}}$ by column have been proposed. These allocate only vectors of size $|\mathcal{S}|$, but require a more complex indexing scheme that maps a state $\mathbf{i}$ to the index $i = \Psi(\mathbf{i}) \in \{0, \ldots, |\mathcal{S}| - 1\}$. Furthermore, in the case of access by columns, for a given reachable destination state $\mathbf{j}$, a source state $\mathbf{i}$ may be unreachable, in which case $\Psi(\mathbf{i})$ must return a null value, to signal that the entry $\widehat{\mathbf{R}}[\mathbf{i}, \mathbf{j}]$ is to be ignored. The lexicographic order mentioned in Sect. 3.1 is an excellent candidate to use for Ψ, but, if used in a simplistic way, it involves an overhead factor $O(\log |\mathcal{S}|)$, since each state $\mathbf{j}$ must be searched in $\mathcal{S}$ to determine its index $\Psi(\mathbf{j})$. Again, the best algorithm appears to be the one using interleaving of the $\mathbf{i}$ and $\mathbf{j}$ indices to maximize the amortization of these searches; when combined with the multi-level data structure of [19] or, even better, the MDDs of [16,17,33], the overhead

is only $O(\log n_1)$. For a thorough discussion of algorithms based on either the potential or the actual state space, see [10].

Matrix Diagrams. In [20], we introduced an alternative method to encode the transition rate matrix $\mathbf{R}$ of a model structured into K sub-models. While related to the expression of Eq. 5, this new method has several advantages over a straightforward Kronecker encoding that uses $(L+1)K$ matrices.

The definition of matrix diagrams is quite similar to that of MDDs, but there are several fundamental differences. First, the nodes of a matrix diagram are matrices, not arrays, since we are encoding a two-dimensional matrix, not a set. Second, the entries of the matrices are lists of pairs, each pair containing a real number and a pointer to a node at the next level, since we are encoding a real value along a path, not just the existence of a path. Finally, as defined, matrix diagrams are not canonical; this is not a problem since the only operation we need to perform on them when solving the CTMC is a vector-matrix multiplication.

Formally, a (nonzero, reduced) *matrix diagram* is a directed acyclic multigraph where:

- Nodes are organized into K *levels*. We write $\langle k.p \rangle$ to denote a generic node, where k is the level and p is a unique index for the nodes at that level. Level K contains only a single node $\langle K.r \rangle$, the *root*, whereas levels $K-1$ through 1 contain one or more nodes reachable on a path from the root.
- A node $\langle k.p \rangle$ is a $n_k \times n_k$ matrix; for $K \geq p > 1$, the entry $\langle k.p \rangle[i,j]$ is a list of pairs of the form (v,q), where v is a real number and $\langle k-1.q \rangle$ is a node, while, for $k=1$, the entry is just a real number v.
- No two elements of a list $\langle k.p \rangle[i,j]$ can have the same value or pointer.
- No two nodes at the same level *duplicate* (i.e., have the same pattern of entries) each other.

Such a data structure can encode a real matrix as follows: the value of the entry in position $(\mathbf{i},\mathbf{j})$ is given by the sum over all paths of the products of the form $v_K \cdots v_1$, where v_k is a real value found in the list in position $(\mathbf{i}_k,\mathbf{j}_k)$ of the matrix at level k for the path. For example, consider the matrix diagram shown in Fig. 9 on the left, and the corresponding matrix encoded by it, on the right; the value of the entry in position $((0,1,1),(1,1,0))$ is given by the product of the entries found in position $(0,1)$ of the one matrix at level 3, in position $(1,1)$ of the first matrix at level 2, and in position $(1,0)$ of the second matrix at level 1: $45 = 1 \cdot 5 \cdot 9$ (only one path needs to be considered because each of the lists corresponding to the row and column indices contains only one entry); the value of the entry in position $((1,1,1),(2,1,1))$ is instead obtained by summing the products corresponding to two paths: $22 = 2 \cdot 2 \cdot 5 + 2 \cdot 1 \cdot 1$.

In [20] we showed several important advantages of matrix diagrams over a traditional matrix-based Kronecker representation:

- A single data structure encodes the entire matrix, unlike the traditional representation which relies on summing, for each event, the Kronecker product of K matrices. This saves a fair amount of indexing overhead.

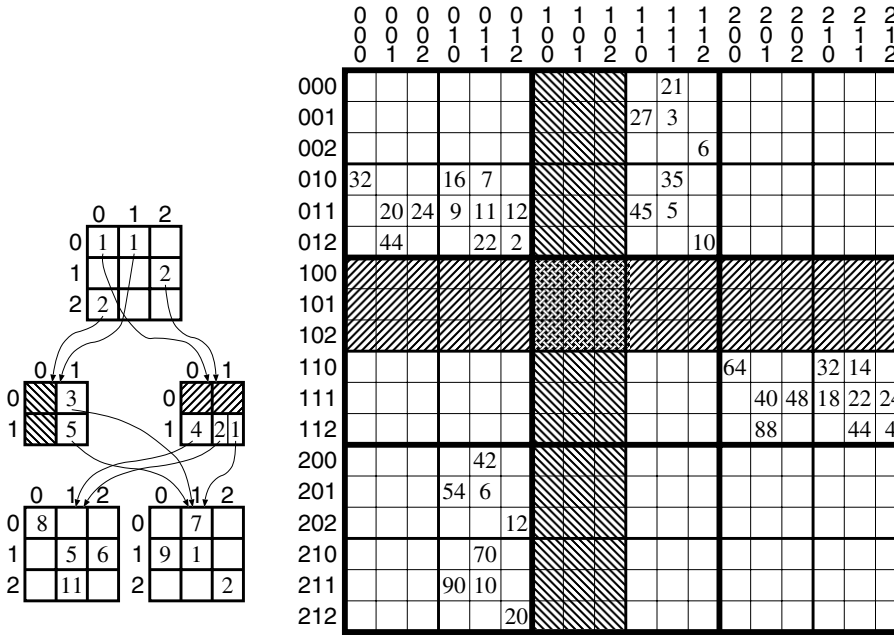


Fig. 9. An example of matrix diagram and the matrix encoded by it.

- If the row and column index sets of the matrix being represented are strict subsets of the cross-products of the row and column index sets at each level (as it is often the case for $\mathbf{R}$ vs. $\widehat{\mathbf{R}}$), a matrix diagram can encode exactly the intended matrix. For example, in Fig. 9, any state of the form $(1, 0, \mathbf{i}_1)$ is unreachable, as indicated by the greyed portions in the large matrix; this is reflected, in the matrix diagram, by the greyed portions in the matrices at level 2: indeed, these can be stored as a 2×1 and a 1×2 matrix, respectively, provided the information about their row and column index set is preserved. This can save both memory and, especially, execution time, since it allows us to operate with "actual" vectors while having the same low overhead enjoyed when using "potential" vectors.
- Operation caches can be used to avoid recomputing partial multiplications of the real values encountered along a path during an iteration of the numerical solution method.
- Finally, because of the ability to represent the desired matrix and not a supermatrix of it, matrix diagrams allow the numerical methods to perform a by-column access without having to worry about the spurious nonzero entries in $\widehat{\mathbf{R}}[\widehat{\mathcal{S}} \setminus \mathcal{S}, \mathcal{S}]$.

Thanks to the above advantages, experimental results show that a Kronecker implementation based on matrix diagrams has a very low overhead both in terms of memory and execution time. For example, in [20], we solved models with tens

of millions of states and hundreds of millions of nonzero entries on a simple Pentium workstation, while explicit solutions were restricted to models about one order of magnitude smaller, and traditional matrix-based Kronecker implementations required either two or three times as much time when using Gauss-Seidel (due to the overhead in by-column access), or approximately twice memory when using Jacobi (due to the need to store both the "old" and the "new" iterates).

6 Conclusion

Formal mathematical models for both logic and temporal analysis of nondeterministic systems are often the most desirable tools, but their solution can be infeasible due to time and, especially, memory limitations.

Parallel and distributed algorithms will continue to play a role in helping to cope with the state-space explosion problem, but they can at best increase the size of the state spaces that can be tackled linearly in the number of resources (processors, memory) we are willing to employ for the solution process.

A more fundamental paradigm shift is achieved using implicit methods based on decision diagrams, Kronecker algebra, and similar approaches that exploit the complex symmetries often present in systems exhibiting globally-asynchronous locally-synchronous behavior. The success of model checking is proof of their effectiveness for the logic modeling aspects but, for Markov models, these approaches are only now beginning to arise the enjoy widespread acceptance.

Our prediction for the area of logic and stochastic models is that an important future research direction will focus on the use of distributed algorithms based on decision diagrams and implicit approaches in general, combined with approximations that avoid the memory bottleneck due to explicit vectors when computing the numerical solution of the underlying stochastic processes.

References

1. S. C. Allmaier, M. Kowarschik, and G. Horton. State space construction and steady-state solution of GSPNs on a shared-memory multiprocessor. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 112–121, St. Malo, France, June 1997. IEEE Comp. Soc. Press.
2. V. Amoia, G. De Micheli, and M. Santomauro. Computer-oriented formulation of transition-rate matrices via Kronecker algebra. *IEEE Trans. Rel.*, 30:123–132, June 1981.
3. J. A. Bergstra, A. Ponse, and S. A. Smolka. *Handbook of Process Algebra*. North-Holland, 2001.
4. J. W. Brewer. Kronecker products and matrix calculus in system theory. *IEEE Trans. Circ. and Syst.*, CAS-25:772–781, Sept. 1978.
5. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comp.*, 35(8):677–691, Aug. 1986.
6. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comp. Surv.*, 24(3):393–318, 1992.

7. P. Buchholz. Numerical solution methods based on structured descriptions of Markovian models. In G. Balbo and G. Serazzi, editors, *Computer performance evaluation*, pages 251–267. Elsevier Science Publishers B.V. (North-Holland), 1991.
8. P. Buchholz. Hierarchical Markovian models – Symmetries and Reduction. In *Modelling Techniques and Tools for Computer Performance Evaluation*. Elsevier Science Publishers B.V. (North-Holland), 1992.
9. P. Buchholz. A class of hierarchical queueing networks and their analysis. *Queueing Systems.*, 15:59–80, 1994.
10. P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS J. Comp.*, 12(3):203–222, Summer 2000.
11. P. Buchholz, M. Fischer, and P. Kemper. Distributed steady state analysis using Kronecker algebra. In B. Plateau, W. J. Stewart, and M. Silva, editors, *Numerical Solution of Markov Chains*, pages 76–95. Prensas Universitarias de Zaragoza, Zaragoza, Spain, Sept. 1999.
12. P. Buchholz and P. Kemper. Numerical analysis of stochastic marked graphs. In *Proc. 6th Int. Workshop on Petri Nets and Performance Models (PNPM'95)*, pages 32–41, Durham, NC, Oct. 1995. IEEE Comp. Soc. Press.
13. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, Pennsylvania, 4–7 June 1990. IEEE Computer Society Press.
14. S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995 (Proc. 16th Int. Conf. on Applications and Theory of Petri Nets, Turin, Italy)*, Lecture Notes in Computer Science 935, pages 181–200. Springer-Verlag, June 1995.
15. G. Ciardo, J. Gluckman, and D. Nicol. Distributed state-space generation of discrete-state stochastic models. *INFORMS J. Comp.*, 10(1):82–93, 1998.
16. G. Ciardo, G. Luettgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000 (Proc. 21th Int. Conf. on Applications and Theory of Petri Nets, Aarhus, Denmark)*, Lecture Notes in Computer Science 1825, pages 103–122. Springer-Verlag, June 2000.
17. G. Ciardo, G. Luettgen, and R. Siminiceanu. Saturation: an efficient iteration strategy for symbolic state space generation. In T. Margaria and W. Yi, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer-Verlag, Apr. 2001. To appear.
18. G. Ciardo and A. S. Miner. SMART: Simulation and Markovian Analyzer for Reliability and Timing. In *Proc. IEEE International Computer Performance and Dependability Symposium (IPDS'96)*, page 60, Urbana-Champaign, IL, USA, Sept. 1996. IEEE Comp. Soc. Press.
19. G. Ciardo and A. S. Miner. Storage alternatives for large structured state spaces. In R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, editors, *Proc. 9th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Lecture Notes in Computer Science 1245, pages 44–57, St. Malo, France, June 1997. Springer-Verlag.
20. G. Ciardo and A. S. Miner. A data structure for the efficient Kronecker solution of GSPNs. In P. Buchholz, editor, *Proc. 8th Int. Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 22–31, Zaragoza, Spain, Sept. 1999. IEEE Comp. Soc. Press.

21. G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. ICASE Report 96-35, Institute for Computer Applications in Science and Engineering, Hampton, VA, May 1996.
22. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
23. M. Davio. Kronecker products and shuffle algebra. *IEEE Trans. Comp.*, C-30:116–125, Feb. 1981.
24. E. W. Dijkstra, W. H. Feijen, and A. Van Gasteren. Derivation of a termination detection algorithm for a distributed computation. *Inf. Proc. Letters*, 16:217–219, June 1983.
25. S. Donatelli. Superposed Stochastic Automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Perf. Eval.*, 18:21–26, 1993.
26. P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplication in stochastic automata networks. *Journal of the ACM*, 45(3):381–414, 1998.
27. A. Graham. *Kronecker Products and Matrix Calculus: with Applications*. Halsted Press / John Wiley, New York, 1981.
28. B. R. Haverkort, A. Bell, and H. Bohnenkamp. On the efficient sequential and distributed generation of very large Markov chains from stochastic Petri nets. In P. Buchholz, editor, *Proc. 8th Int. Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 12–21, Zaragoza, Spain, Sept. 1999. IEEE Comp. Soc. Press.
29. T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.
30. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL: Status and developments. In O. Grumberg, editor, *9th Int. Conf. on Computer Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 456–459. Springer-Verlag, June 1997.
31. P. Marenzoni, S. Caselli, and G. Conte. Analysis of large GSPN models: a distributed solution tool. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 122–131, St. Malo, France, June 1997. IEEE Comp. Soc. Press.
32. V. Migallón, J. Penadés, and D. B. Szyld. Experimental study of parallel iterative solutions of Markov chains with block partitions. In B. Plateau, W. J. Stewart, and M. Silva, editors, *Numerical Solution of Markov Chains*, pages 96–110. Prensas Universitarias de Zaragoza, Zaragoza, Spain, Sept. 1999.
33. A. S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In H. Kleijn and S. Donatelli, editors, *Application and Theory of Petri Nets 1999 (Proc. 20th Int. Conf. on Applications and Theory of Petri Nets, Williamsburg, VA, USA)*, Lecture Notes in Computer Science 1639, pages 6–25. Springer-Verlag, June 1999.
34. T. Murata. Petri Nets: properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–579, Apr. 1989.
35. D. Nicol and G. Ciardo. Automated parallelization of discrete state-space generation. *J. Par. and Distr. Comp.*, 47:153–167, 1997.
36. D. M. Nicol. Non-commital barrier synchronization. *Parallel Computing*, 21:529–549, 1995.
37. E. Pastor and J. Cortadella. Efficient encoding schemes for symbolic analysis of Petri nets. In *Proc. Design Automation and Test in Europe*, Feb. 1998.
38. E. Pastor and J. Cortadella. Structural methods applied to the symbolic analysis of Petri nets. In *Proc. IEEE/ACM International Workshop on Logic Synthesis*, June 1998.

39. E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri net analysis using boolean manipulation. In R. Valette, editor, *Application and Theory of Petri Nets 1994, (Proc. 15th Int. Conf. on Applications and Theory of Petri Nets, Zaragoza, Spain)*, Lecture Notes in Computer Science 815, pages 416–435. Springer-Verlag, June 1994.
40. S. Pissanetzky. *Sparse Matrix Technology.* Academic Press, 1984.
41. B. Plateau. On the stochastic structure of parallelism and synchronisation models for distributed algorithms. In *Proc. 1985 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pages 147–153, Austin, TX, USA, May 1985.
42. W. J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. *Eur. J. of Oper. Res.*, 86:503–525, 1995.
43. K. Varpaaniemi, J. Halme, K. Hiekkanen, and T. Pyssysalo. PROD reference manual. Technical Report B13, Helsinki Univ. of Technology, 1995.

A A Brief Introduction to Kronecker Operators

Given K real matrices $\mathbf{A}_k \in \mathbb{R}^{n_k \times n_k}$, for $K \geq k \geq 1$, their *Kronecker product*

$$\mathbf{A} = \mathbf{A}_K \otimes \mathbf{A}_{K-1} \otimes \cdots \otimes \mathbf{A}_1 = \bigotimes_{K \geq k \geq 1} \mathbf{A}_k \in \mathbb{R}^{n_{K:1} \times n_{K:1}}$$

is defined by

$$\mathbf{A}[\mathbf{i}, \mathbf{j}] = \mathbf{A}_K[\mathbf{i}_K, \mathbf{j}_K] \cdots \mathbf{A}_K[\mathbf{i}_1, \mathbf{j}_1],$$

where we use the mixed-base indexing scheme

$$\mathbf{i} \equiv (\mathbf{i}_K, \ldots, \mathbf{i}_1) = (\ldots((\mathbf{i}_K) \cdot n_{K-1} + \mathbf{i}_{K-1}) \cdot n_{K-2} \cdots) \cdot n_1 + \mathbf{i}_1 = \sum_{K \geq k \geq 1} \mathbf{i}_k \cdot n_{k-1:1},$$

while the *Kronecker sum* of the same matrices is simply

$$\bigoplus_{K \geq k \geq 1} \mathbf{A}_k = \sum_{K \geq k \geq 1} \mathbf{I}_{n_{K:k+1}} \otimes \mathbf{A}_k \otimes \mathbf{I}_{n_{k-1:1}} \in \mathbb{R}^{n_{K:1} \times n_{K:1}}.$$

For example, given $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} r & s & t \\ u & v & w \\ x & y & z \end{bmatrix}$, we have

$$\mathbf{A} \otimes \mathbf{B} = \left[\begin{array}{c|c} a\mathbf{B} & b\mathbf{B} \\ \hline c\mathbf{B} & d\mathbf{B} \end{array}\right] = \left[\begin{array}{ccc|ccc} ar & as & at & br & bs & bt \\ au & av & aw & bu & bv & bw \\ ax & ay & az & bx & by & bz \\ \hline cr & cs & ct & dr & ds & dt \\ cu & cv & cw & du & dv & dw \\ cx & cy & cz & dx & dy & dz \end{array}\right] \quad \text{and}$$

$$\mathbf{A} \oplus \mathbf{B} = \left[\begin{array}{ccc|ccc} a & & & b & & \\ & a & & & b & \\ & & a & & & b \\ \hline c & & & d & & \\ & c & & & d & \\ & & c & & & d \end{array}\right] + \left[\begin{array}{ccc|ccc} r & s & t & & & \\ u & v & w & & & \\ x & y & z & & & \\ \hline & & & r & s & t \\ & & & u & v & w \\ & & & x & y & z \end{array}\right] = \left[\begin{array}{ccc|ccc} a{+}r & s & t & b & & \\ u & a{+}v & w & & b & \\ x & y & a{+}z & & & b \\ \hline c & & & d{+}r & s & t \\ & c & & u & d{+}v & w \\ & & c & x & y & d{+}z \end{array}\right].$$

Intuitively, the Kronecker product expresses *contemporaneous* or *synchronized* actions: if $\mathbf{A}$ and $\mathbf{B}$ are the transition probability matrices of two independent DTMCs, $\mathbf{A} \otimes \mathbf{B}$ is the transition probability matrix of their composition, while the Kronecker sum expresses *asynchronous* behavior: if $\mathbf{A}$ and $\mathbf{B}$ are the infinitesimal generator matrices of two independent CTMCs, $\mathbf{A} \oplus \mathbf{B}$ is the infinitesimal generator matrix of their composition.

Note that the definition of Kronecker product (but not that of Kronecker sum) can be extended to rectangular matrices, but we only need it for square matrices in our case.

General Distributions in Process Algebra

Joost-Pieter Katoen and Pedro R. D'Argenio

Formal Methods and Tools Group, Dept. of Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract. This paper is an informal tutorial on stochastic process algebras, i.e., process calculi where action occurrences may be subject to a delay that is governed by a (mostly continuous) random variable. Whereas most stochastic process algebras consider delays determined by negative exponential distributions, this tutorial is concerned with the integration of *general, non-exponential* distributions into a process algebraic setting. We discuss the issue of incorporating such distributions in an interleaving semantics, and present some existing solutions to this problem. In particular, we present a process algebra for the specification of stochastic discrete-event systems modeled as generalized semi-Markov chains (GSMCs). Using this language stochastic discrete-event systems can be described in an abstract and modular way. The operational semantics of this process algebra is given in terms of stochastic automata, a novel mixture of timed automata and GSMCs. We show that GSMCs are a proper subset of stochastic automata, discuss various notions of equivalence, present congruence results, treat equational reasoning, and argue how an expansion law in the process algebra can be obtained. As a case study, we specify the root contention phase within the standardized IEEE 1394 serial bus protocol and study the delay until root contention resolution. An overview of related work on general distributions in process algebra and a discussion of trends and future work complete this tutorial.

1 Introduction

The design and analysis of systems, like embedded systems, communication protocols or multi-media systems, requires insight in not only the functional, but also in the real-time and performance aspects of applications involved. Researchers in formal methods (i.e., concurrency theory) have recognized the need for the additional support of quantitative aspects, and various initiatives have been taken to accomplish such support. A prominent example is the treatment of real-time constraints, where specification formalisms like timed automata [2] have emerged, and impressive progress has been made in the development of efficient verification algorithms [15,72]. This has resulted in a number of tools (model checkers) that provide interesting experimental platforms for industrial case studies.

Hard and soft real-time constraints. Constraints that one typically considers in this real-time setting are 'hard', for instance,

"the system must always do a certain activity before time t"

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 375–429, 2001.

For many applications, though, real-time constraints are typically less stringent. Rather than requiring that certain activities *must always* occur before time t, in practice one is usually interested in more 'soft' real-time constraints, where a system is required to perform the activity *mostly* before t. The soft real-time requirements of systems typically address their performance characteristics, and are often also referred to as their quality-of-service (QoS) parameters. They are usually related to stochastic aspects of various forms of time delay, such as, for example, mean and variance of message transfer delay, service waiting times, failure rates, utilization, etc. In a soft real-time system one typically considers constraints like:

"the system should perform an activity before time t in 92% of the cases"

In soft real-time systems, state changes take place in a discrete fashion and the time of occurrence of activities is controlled by random variables. These systems are also known as *stochastic discrete-event simulation* (DES) models. In contrast to most formalisms that are restricted to a particular set of probability distributions, like negative exponential or discrete distributions, the objective is to support *general* distributions, discrete or continuous. This makes the formalism more expressive and more interesting from a practical point of view.

The need for a single framework. Traditionally, there has been a clear separation between the functional and performance aspects of systems, and as a result different communities have constructed and analyzed their own, largely unrelated models for the aspects under their responsibility. This has resulted in what has been recognized as "the insularity problem of performance evaluation in the system design process" [47]. In modern systems, though, the difference between functional and performance features has become blurred, and both features are becoming of comparable interest. Thus, it would be beneficial to be able to check how changes in functionality affect performance issues, and vice versa. In addition, one would like to have a better relationship between the models that are used for qualitative and quantitative analysis, and avoid the use of different models that are mutually incompatible. A single framework where both aspects could be defined would be highly desirable [24,37]. This tutorial is focused on an integrated approach using *process algebra*.

1.1 Organization of the Paper

Section 2 contains an introduction on stochastic process algebra (to what extent do they differ from traditional process algebra?), justifies the usage of general distributions (why do we need them?), sketches the complications of incorporating general distributions in process algebra (why are things not so straightforward as for exponential distributions?), and provides an overview of possible solutions that have been suggested so far. Section 3 introduces generalized semi-Markov processes (GSMPs), a model for general distributions. Section 4 presents a couple of small examples that serve as a justification for providing a process algebraic framework for this model. Section 5 presents stochastic automata, an extension

of labelled transition systems that we use as a semantic model of our process algebra with general distributions, called SPADES (Stochastic Process Algebra for Discrete-Event Simulation) and symbolized as  . Section 7 gives an account of evaluation techniques for  specifications, including quantitative methods – discrete-event simulation – and qualitative methods – checking of (timed) safety properties. Section 9 discusses related work on process algebras with general distributions. Section 10 provides a summary of the tutorial and presents some topics for further research.

2 Fitting General Distributions in Process Algebra

In traditional process algebras, like ACP [39], CCS [77], CSP [58] and LOTOS [64], a (possibly concurrent) system is syntactically represented using powerful composition operators which facilitate the development of modular and well-structured specifications. The formal meaning of a process algebra term is defined in a mathematical model. By defining an appropriate equivalence relation on this model one is able to formally compare and transform (e.g., simplify or reduce) specifications. If, in addition, this relation is a congruence[1], then such transformation can be carried out component by component. This compositional nature reduces the complexity of the transformation significantly. Finally, due to the algebraic nature of the formalism it is possible to define equational rules on the syntax that allow to perform step-wise design and minimization at a purely syntactic level, without any reasoning in semantic terms.

2.1 Stochastic Process Algebra

Traditionally, process algebras have concentrated on the functional aspects of systems such as their observable behavior, control flow and synchronization as properties in relative time. In the late eighties, the interest grew in extending process algebras with quantitative information like time and (discrete) probabilities. These extensions are known as timed and probabilistic process algebras, respectively.

Timed process algebra. Timed extensions of ACP [5], CCS [78,99], CSP [89] and LOTOS [13,73] have been defined. The basic idea underlying timed process algebras is to change the role of action-prefix, denoted by $a;p$ for action a and process p. Originally, the expression $a;p$ simply means that first an action a is offered, and after the appearance of a the process behaves like p. No statement is made about when action a occurs. In timed process algebra there are basically two schools:

- replace $a;p$ by $(a,t);p$ denoting that action a is offered after a delay of t time units, or

[1] An equivalence relation is a congruence if two equivalent terms behave indistinguishable in any context.

– extend the language with a timed prefix like $t \mapsto p$ denoting that process p is reached after a delay of t time units; $t \mapsto a; p$ means that action a is offered after t time units.

(There are several finer points that we ignore here; see [80] for an overview.) The last distinction leads to a behavior where two distinct phases are separated. Phases, during which one or more actions occur together with their corresponding state changes, but where no time elapses, are distinguished from phases where time passes, but during which no actions happen. With some appropriate modifications, timed process algebras can be considered as high-level specification formalisms for timed automata [29].

Probabilistic process algebra. Probabilistic extensions of ACP [6], CCS [44], CSP [75] and LOTOS [76] have been studied. A recent overview of probabilistic process algebras can be found in [65]. The basic idea of these calculi is to incorporate a probabilistic choice operator that allows terms like $p \oplus_\pi q$ (with $\pi \in (0,1)$) where process p can be selected with probability π and process q with $1-\pi$. Different semantic models have been used for probabilistic process calculi, depending on whether non-determinism is allowed or not. In the deterministic case, these languages represent discrete-time Markov chains (DTMC) [68]; in presence of non-determinism, models similar to Markov decision processes [34] are obtained.

Markovian process algebra. In stochastic process algebras, time and probability are integrated by considering delays of a continuous probabilistic nature. In languages like EMPA [9], PEPA [56] or TIPP [43], a non-negative real-valued rate is associated to actions that determines probabilistically the delay prior to an action. For rate λ, the term $(a, \lambda); p$ denotes that action a is offered after a delay determined by a negative exponential distribution. More precisely, $(a, \lambda); p$ offers action a within t time units with probability $1 - e^{-\lambda t}$ and then evolves into process p. The mean duration until action a is offered is thus $1/\lambda$. As a semantic model, transition systems are used where transitions are labelled with pairs of actions and rates. By omitting the action labels — but keeping the rate information — one obtains a (time-homogeneous) continuous-time Markov chain (CTMC) for which steady-state and transient performance metrics can be obtained using traditional techniques [95]. These process algebras provide a high-level specification formalism for CTMCs. Due to this property they are also called *Markovian* process algebras; recent overviews can be found in [19,50,57].

Separating delays and actions. The major distinction between Markovian process algebras is the treatment of time consumption in case of interaction. Technically, this amounts to the computation of the resulting rate in case two actions like (a, λ) and (a, μ) synchronize. To our opinion, the most natural interpretation is to require both delays to have completed before the synchronization (on a) can take place — the so-called *patient communication* [55]. The thus resulting random variable equals the maximum of the random variables that are exponentially distributed with rates λ and μ, respectively. This random variable is, however, not exponentially distributed. To overcome this technical problem, several so-

lutions have been suggested that either lack a clear stochastic interpretation or have a somewhat restricted applicability.

The stochastic interpretation (i.e., maximum of random variables) can be obtained in a rather natural way by explicitly separating the advance of time and the occurrence of actions. In this way, synchronization only takes place via immediate actions. Thus, the usual prefix $a;p$ remains unchanged, but is complemented with a delay prefix $\lambda \mapsto p$ which evolves into p after an exponential delay with mean $1/\lambda$. This separation of discrete and continuous phases is similar to that in some timed process algebras (see before) and has been proposed in the context of Markovian process algebras in [49,51,52].

2.2 Beyond Exponential Distributions

Integration of general, non-exponential distributions in a process algebraic setting has received scant attention. Instead, most work has been focussed on exponential distributions. Although exponential distributions yield analytically tractable models (i.e., CTMCs), and are useful for many applications, they are not realistic for modeling many phenomena in an adequate way. For example:

- in performance modeling, it is often convenient to incorporate empirical distributions into the model that have been obtained by measuring a realization of the system. These measurements may e.g., indicate the traffic intensity of a communication network at a working day, or indicate the length of communicated web pages during peak hours. These distributions are mostly not exponential.
- if a distribution function G is only partially known, it is preferably approximated by a probability distribution F with a "maximal indeterminacy" in the sense that it is impossible to recognize from F any preference of one event over another. Thus, F assumes the least about the structure of the distribution G, or, stated otherwise, it has the highest degree of randomness. Technically speaking, F maximizes the *entropy* [93]:

$$-\int_{-\infty}^{\infty} f(x)\ln(f(x))\,dx$$

  where f is the probability density function of F. Depending on which partial information about G is available, different appropriate choices for F remain. If only the mean and variance are known, the normal distribution is the most indeterminate; in cases where only the minimum and maximum are known, the uniform distribution on the interval between these bounds is the most indeterminate. The exponential distribution is the most indeterminate approximation only in cases where only the mean is known (of a positive random variable).
- empirical studies have shown that many system parameters, such as sizes of data files stored on web servers and transferred through the Internet, job service times in general-purpose computing environments, and node degrees

of certain graph structures (such as hyper links of web pages), exhibit so-called *heavy-tail* distributions, i.e., distributions with a very high variance. A distribution F is heavy tailed [26] if for positive constant c:

$$F(x) \text{ "approximates" } 1 - c \cdot x^{-\alpha} \text{ for } 0 < \alpha < 2$$

F has an infinite variance, and for $\alpha < 1$ it has an infinite mean. An important heavy-tail distribution is the Pareto distribution. For a heavy-tail distribution (with α=1), about 60% of the probability mass is contained in just 1% of the observations; for an exponential distribution this dependency is roughly linear. If one observes heavy-tailed inter-arrivals, then the longer one has waited, the longer we should expect to wait — the expectation paradox. Instead, for exponential distributions the waiting time does not play any role, due to the memoryless property (see next Section).

- deterministic delays are prevalent and important in computer, communication and manufacturing systems. Typical examples of deterministically distributed parameters are: timeouts in communication protocols, hard deadlines in real-time systems, transmission delays of fixed-length packets, and cycle times of work-flow management systems.
- it has been argued that several phenomena in modern communication systems, in particular several aspects of multi-media communication systems, can be most adequately modeled by non-exponential distributions. For instance, the variability of the delay of sound and video frames (so-called jitter) is mostly assumed to be controlled by a normal distribution [11,38].

2.3 Interleaving + General Distributions = Non-trivial

Given the need for non-exponential distributions, the question is whether we cannot simply replace the exponential distributions in Markovian process algebra by general distributions. This turns out not to be straightforward. We illustrate this by discussing (a simplified version of) an important axiom in process algebra, known as the expansion law.

Expansion law. A popular mathematical model for providing semantics to traditional process algebras is labelled transition systems, (possibly infinite-state) automata where transitions that are labelled with actions describe how the system can evolve from one state to another. In mapping process algebra terms to this model, the independent parallel composition of two actions is treated as a choice between the two possible sequential orderings. Thus, in a parallel composition, actions of one component are interleaved with actions of the other — hence the term *interleaving* semantics. As a result, independent parallel composition (denoted $||$) can be reduced in terms of choice (denoted $+$) and prefix as expressed by, for example:

$$a; p \,||\, b; q = a; (p \,||\, b; q) + b; (a; p \,||\, q) \tag{1}$$

for actions a, b and processes p, q (cf. Fig. 1 for p and q being the process $\mathbf{0}$ that cannot perform any action). This principle, in its full generality known as the

expansion law [77], is widely accepted and has proven to be of crucial importance for process algebraic verification purposes [4].

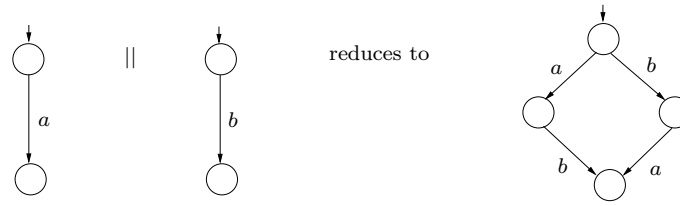


Fig. 1. Interleaving of the processes $a;\mathbf{0}$ and $b;\mathbf{0}$

Exponential distributions and interleaving semantics fit well. The semantics of Markovian process algebras are commonly defined using an extension of labelled transition systems. The structure of these transition systems closely resembles that of CTMCs. The elegant memoryless property of exponential distributions enables a smooth integration in an interleaving setting, since in analogy of (1) we have:

$$\lambda \mapsto p \,||\, \mu \mapsto q = \lambda \mapsto (p \,||\, \mu \mapsto q) + \mu \mapsto (\lambda \mapsto p \,||\, q) \tag{2}$$

To justify this law, consider the term $\lambda \mapsto p \,||\, \mu \mapsto q$. Let U be the random variable modeling the delay before process q can start — U is thus exponentially distributed with rate μ — and suppose that the delay of the left process "finishes first" (with rate λ), say, after y time units. The probability that the start of process q has to delay for at most an additional x time units is

$$\Pr[U \leqslant y{+}x \mid U > y]$$

Due to the memoryless property of exponential distributions, it holds that

$$\Pr[U \leqslant y{+}x \mid U > y] = \Pr[U \leqslant x] \tag{3}$$

Thus, the remaining duration until the initial delay of process q finishes is (again) determined by an exponential distribution with rate μ. Stated differently, the delay of the left process does not have any impact on the distribution of the remaining delay in the other process — the advance of time governed by memoryless distributions is independent. By symmetry, an analogous reasoning applies when the right-hand process finishes first.

General distributions and expansion law. If we allow actions to be delayed by general distributions F and G, though, it turns out that the analogon of (2) is invalid:

$$F \mapsto p \,||\, G \mapsto q \neq F \mapsto (p \,||\, G \mapsto q) + G \mapsto (F \mapsto p \,||\, q) \tag{4}$$

The reason for this inequality is the absence of the memoryless property for general distributions. For instance, after the delay imposed by F in the left-hand process, the residual delay of the right-hand process $G \mapsto q$ has to be

taken into account in order to correctly determine the remaining delay before process q becomes enabled.

2.4 Some Solutions to the Problem

If the incorporation of general distributions into process algebra is not trivial, what are possible strategies to overcome this problem? Here, we summarize the main schools of thought.

Still use Markovian process algebras. In this category we find two kinds of solutions: approximation and exploiting insensitivity.

1. A possible solution is to *approximate* general distributions by appropriate probability distributions that can be described as series/parallel combinations of exponential distributions, possibly with feedback, thus residing in the class of Markovian process algebras. An interesting class of distributions for this purpose is the class of phase-type (PH) distributions [79]. They are defined as distributions of absorption times in finite CTMCs (but that may contain loops) with a single absorbing state, i.e., a state without any outgoing transition. PH-distributions can approximate any distribution on $[0, \infty)$ arbitrarily close; algorithms to fit a PH-distribution to empirical distributions do exist [3]. The encoding of PH-distributions in a Markovian process algebra has been considered in [49,51,52] where — due to the aforementioned separation between time and actions — any CTMC with a trivial initial distribution (and thus any such PH-distribution) can be specified. A similar approach can be taken by using a subset of PH-distributions such as Cox [25] or Erlang mixture distributions, see [52] and [23], respectively.
2. An alternative solution is to allow general distributions in a controlled way such that the stochastic property of *insensitivity* can be exploited. A stochastic process is insensitive if its steady-state distribution depends on the distribution of one or more of the random variables governing state residence times only through their mean. The theory of insensitivity has been applied to high-level specification formalisms such as stochastic Petri nets [48]. In [22,23] a syntactic construction is presented that guarantees the insensitivity of the stochastic process underlying a stochastic process algebra specification. By means of this construction it is guaranteed that for studying the steady-state behavior of the process under consideration, it is sufficient to consider that process in which each general distribution is replaced by, for instance, an exponential distribution with the same mean. Hence, the steady-state behavior of such processes can be analyzed using ordinary CTMC techniques.

The main benefit of both approaches is that existing frameworks can be used without any modification. The main disadvantage of approximation (approach 1) is that it leads to a state-space explosion since any general distribution is represented by a (possibly complex) CTMC. Besides, choice expressions like $F \mapsto p + G \mapsto q$ are difficult to treat as the choice between the approximations of F and G is determined by the first phase of their PH-distribution, and not

by their entire PH-distribution. The insensitivity approach (2) is applicable to a small class of processes and is restricted to steady-state properties.

Drop the expansion law. As we have seen before, obtaining an expansion law (11) is a serious problem when considering general distributions. A feasible option is to define a semantics for process algebra that does not obey this law. The thus obtained semantics is known as *true concurrency* or *partial-order* semantics. In these models, system runs are no longer represented as totally ordered sequences, but rather as partial orders where the occurrences of causally independent actions are unrelated. Important partial-order models include Petri nets [36], and event structures [81]. The idea of using true concurrency semantics for stochastic process algebra has been brought up in [18] where a stochastic variant of event structures was used as semantic model. As actions that occur concurrently are unordered, there is no need to keep track of the residual delay of random variables that "run in parallel". Analysis techniques that have considered for these event structures are discrete-event simulation [67], and a decomposition-based analysis method [12]. Similar approaches have been pursued in [83] where causal dependencies between delays are derived from the transition labels (so-called proved transitions) and in [53] where stochastic task graphs are used, a model that is quite similar to event structures.

The main advantage of this approach is the potential compact representation of the state space; the main disadvantage is that it leads to infinite-state semantic objects even for simple recursive terms. Recent investigations indicate, however, that finite objects can be obtained for the event structure approach (for finite-state process algebra terms) from which stochastic task graph models are generated that can be analyzed numerically [87].

Keep track of residual lifetimes. As a third solution, we refine the earlier discussed separation of time and actions a bit further and distinguish between:

- the start of a probabilistic delay,
- the completion of a probabilistic delay, and
- the occurrence of immediate actions.

To keep track of delays labelled transition systems are extended with *clocks*. A clock is initialized by sampling a probability distribution function, and starts counting down once initialized. All clocks count down at the same pace. Transitions are labelled with an action and a set of clocks; this transition is enabled when all clocks in the set have expired, i.e., have reached the value 0. Similarly, we extend traditional process algebra with two new constructs: for C a finite set of clocks, $\mathsf{when}\ C \mapsto p$ denotes the process that after expiration of all clocks in C behaves like p, and $\mathsf{set}\ C\ \mathsf{in}\ p$ denotes the process that behaves like p after any clock x in C is initialized according to its distribution. The delay prefix $\lambda \mapsto p$ that we encountered before is now written as $\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p)$ with x a clock controlled by an exponential distribution of rate λ. Note that in the exponential case the distinction between start and finish of a delay is not needed since

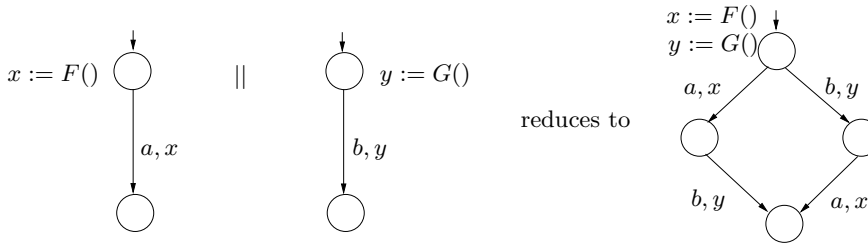


Fig. 2. Interleaving processes set x in (when $x \mapsto a; \mathbf{0}$) and set y in (when $y \mapsto b; \mathbf{0}$)

$$\begin{aligned} &\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; \mathsf{when}\ y \mapsto p) \\ =\ &\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a); \mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p) \end{aligned}$$

When mapping process algebra terms onto the extended labelled transition systems, the principle of interleaving is applied (cf. Fig. 2). In the resulting automaton, initially both clocks x and y are initialized and start counting down. If x expires first, action a happens, and a state is reached in which clock y records the remaining time until action b is enabled. A symmetric scenario happens when clock y expires first. Accordingly, an expansion law can be obtained. For instance, for $p' = \mathsf{when}\ x \mapsto a; p$ and $q' = \mathsf{when}\ y \mapsto b; q$ we have:

$$\mathsf{set}\ x\ \mathsf{in}\ p' \,||\, \mathsf{set}\ y\ \mathsf{in}\ q' = \mathsf{set}\ x, y\ \mathsf{in}\ \ (\mathsf{when}\ x \mapsto a; (p \,||\, q') + \mathsf{when}\ y \mapsto b; (p' \,||\, q))$$

This idea has first been brought up in [30], and has been extended and refined later on [27,31,32,33]. In this tutorial we will follow this direction. A similar approach has been taken in [17] where a "start-termination" (ST) semantics — a model that has been originally developed to study refinement in process algebra [40] — is used for generally distributed delays.

The labelled transition systems extended with clocks closely resemble *generalized semi-Markov processes* (GSMPs), a model used for the study of stochastic discrete-event systems. Accordingly, the process algebra that allows for general distributions can be considered as a high-level specification formalism for GSMPs.

3 Generalised Semi-Markov Chains

GSMPs have been introduced by Glynn [41] and Whitt [100]; for an introduction to GSMPs we refer to [21,92]. We consider discrete-state GSMPs, *generalised semi-Markov chains* (GSMCs, for short).

The model of GSMCs. A GSMCs is a state automaton where transitions are triggered by the occurrence of stochastically timed events. A set of active events is associated to each state. These are the events that are possible in that state, i.e., that can cause a transition outgoing from that state. The remaining time

until the possible occurrence of an event is determined by its clock; we thus have one clock per event. Clocks are initialised according to a continuous probability distribution function and run backwards, all with the same pace. In the following we assume a set of clocks $\mathcal{C}$ is given.

Definition 1. $(Z, z_0, \mathbf{E}, E, C, \mathsf{next})$ *is a* generalised semi-Markov chain *(GSMC) with*[2]

- Z*, a non-empty set of* states *with* initial state $z_0 \in Z$*,*
- $\mathbf{E}$*, a non-empty set of* events*,*
- $E : Z \to \mathcal{P}_f(\mathbf{E})$*, the* event-assignment *function s.t.* $E(z) \neq \varnothing$ *for all* $z \in Z$*,*
- $C : \mathbf{E} \to \mathcal{C}$*, the* clock-assignment *function, with continuous distribution* $F_{C(e)}$ *for any* $e \in E$*,*
- $\mathsf{next} : Z \times \mathbf{E} \to Z$*, the partial* next-state *function that assigns to each state* z *and event* $e \in E(z)$ *a next state* $\mathsf{next}(z, e)$*.*

Example 1. Consider a queueing system in which jobs arrive and wait until they are executed by a single server. An infinite population of jobs is assumed. Jobs arrive with an inter-arrival time that is determined by a continuous probability distribution F while the delay between the processing of two successive jobs is controlled by distribution H. The serving discipline is FCFS (first come first served). This system is known as a $G/G/1/\infty$-queue, where G stands for general distribution of the arrival and service process, respectively, 1 indicates the number of servers, and ∞ denotes the infinite buffer capacity. A typical GSMC description of such queueing system is defined in the following way. We let the state space $Z = \{0, 1, 2, \dots\}$ where the state number indicates the number of jobs that are currently in the system, i.e., in the queue or currently being processed. The initial state z_0 is 0, the empty system. In each state, possible events are the arrival of a job (denoted a) and the completion of a job (denoted c); thus, $\mathbf{E} = \{a, c\}$. In the initial state no job completion is possible. Thus, $E(i) = \{a, c\}$ for $i > 0$ and $E(0) = \{a\}$. The arrival of a job causes a transition from state i to state $i{+}1$. Completion of a job leads to a transition from state $i{+}1$ to state i. Thus, $\mathsf{next}(i, a) = i{+}1$ and $\mathsf{next}(i{+}1, c) = i$. The state-transition structure of this GSMC is depicted in Fig. 3. Clocks are initialised as follows. On entering state $i{+}1$ after an arrival, the clock of the next arrival a is initialised

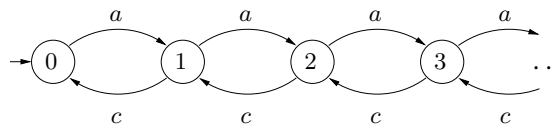


Fig. 3. GSMC for a G/G/1/∞ queueing system

[2] We adopt the following notational convention. For a set A, $\mathcal{P}(A)$ denotes the set of all subsets of A, and $\mathcal{P}_f(A)$ denotes the set of finite subsets of A.

according to distribution F, the job inter-arrival time. On entering state i after a job completion, the clock of the next job completion is initialised according to distribution H, the service delay. Accordingly, we let $C(a) = x$, $C(b) = y$ and $F_x = F$ and $F_y = H$.

The behaviour of a GSMC. The dynamics of a GSMC are described as follows in a procedural way. Note that for any state z, to each active event $e \in E(z)$ a clock value $val(C(e)) \geqslant 0$ is associated. Initially, all active events are initialised according to their distribution function, i.e., $val(C(e)) = F_{C(e)}(\cdot)$ if $e \in E(z_0)$. Intuitively, $F_{C(e)}(\cdot)$ denotes "taking a sample from distribution $F_{C(e)}$". Then:

1. determine the set of active events $E(z)$ in the current state z
2. determine the clock value d^* such that $d^* = \min\{\, val(C(e)) \mid e \in E(z) \,\}$
3. determine event e^* with $val(C(e^*)) = d^*$ and state $z' = \mathsf{next}(z, e^*)$
4. determine the new clock values val' in z' as follows:

$$val'(C(e)) = \begin{cases} val(C(e)) - d^* & \text{if } e \in E(z) \cap (E(z') - \{\, e^* \,\}) \\ F_{C(e)}(\cdot) & \text{if } e \in E(z') - (E(z) - \{\, e^* \,\}) \\ \infty & \text{otherwise} \end{cases}$$

5. go back to the first step of the procedure with current state z'.

Note that there always exists an event e^*, since $E(z)$ is non-empty for every state z. Since all clocks are initialised by continuous distributions, the event e^* is guaranteed to be *unique* with probability one, as the probability of sampling two continuous distributions with the same value is 0. Once event e^* with minimal clock value has been determined, the next state z' is determined (step 3.) and the new clock values are calculated as follows (step 4.). For each active event e in state z that remains active in z', the clock value is decreased by the elapsed time d^*. For each event e that becomes active in z', the clock value is determined by sampling the clock-distribution function $F_{C(e)}$; for all other events the clock value equals ∞, indicating that these events are inactive.

The above procedure is also known as variable time-advance procedure [92] which is characterised by time steps of varying length and an event occurrence in every time step. This procedure is controlled by the occurrence of "next events" and the time between the occurrence of two events is "skipped". This principle is reflected by the fact that clock values do not increase as time passes, but only increase if the next event happens (see step 4.).

Example 2. Fig. 4 presents an example execution of the GSMC of Fig. 3 by depicting the values of the clocks of events a (solid line) and c (dashed line) (on the y-axis) while time evolves (x-axis). Each time a clock expires, a transition to the next state is taken. Below the x-axis, for each time instant the current state is indicated. Note that in this execution, event c becomes enabled when moving from the initial state to state 1 and stays enabled while visiting states 2 and 3, before triggering the transition back to state 2.

Restrictions. Actually, we consider a subclass of GSMCs as introduced in [41]. The main restriction that we impose is that the next state is *deterministically*

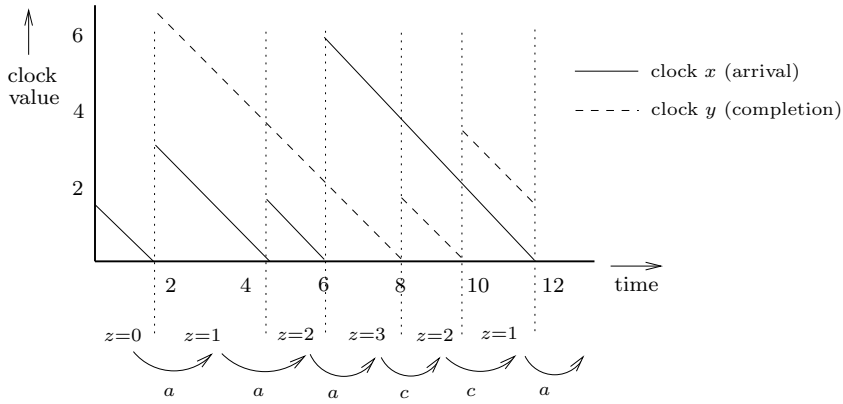


Fig. 4. A sample evolution of the GSMC of Fig. 3

determined by the present state and the triggered event, whereas in the original GSMCs the next state is chosen in a discrete probabilistic fashion from a set of possible next states. In addition, we consider *time-homogeneous* GSMCs. Such processes are invariant under time-shifts. Intuitively, the probability to be in state z' at time t' given that the system is in state z at time $t < t'$, is equal to the probability that the system is in state z' at time $t'-\Delta$ given that the system is in state z at time $t-\Delta$ (for any $\Delta \leqslant t$). This is a rather common restriction. Finally, clocks in GSMCs are allowed to have different (possibly state-dependent) rates whereas in our case all clocks proceed with the same speed. Different rates are not very usual in discrete-event simulation, and moreover, under certain conditions, such multi-rated GSMCs can be represented by GSMCs where all clock rates equal one. The notion of GSMC considered here can thus be summarised as *mono-rated, deterministic, time-homogeneous* GSMCs.

GSMCs versus CTMCs and SMCs. In order to better understand the link with related models, we briefly address the relationship of GSMCs to continuous-time Markov chains (CTMCs) [95] and semi-Markov chains (SMCs) [86,54]. To put it in a nutshell, a CTMC is a GSMC in which all clocks are governed by negative exponential distributions, i.e., for each clock x we have $F_x(t) = 1-e^{-\lambda t}$, for some non-negative real λ. A CTMC possesses the Markov property: the probability of making a transition to a next state only depends on the current state and not on previous states ("absence of state memory"). The memoryless property of exponential distributions further implies that the probabilities of taking next transitions do not depend on the amount of time spent in the current state ("absence of age memory"). The residence time of a state is exponentially distributed with a rate that equals the sum of the rates of its outgoing transitions.

A SMC is a GSMC in which all clocks are initialised on each state change. As exponential distributions are memoryless, cf. equation (3), there is no difference between setting clocks on each state change or not. Each CTMC is thus an SMC. An SMC is, however, not a CTMC. It possesses the Markov property,

but does not conform to the "absence of age memory" principle: probabilities of taking next transitions do depend on the amount of time spent in the current state. For an SMC, the state residence times are generally distributed and are explicitly specified for each state. In a GSMC, the residence time of a state is determined implicitly by the distributions of the set of active events in a state. As a result, the state residence times in a GSMC may be history-dependent. This phenomenon is the essential difference with SMCs, where state residence times are governed by an a priori, fixed random variable.

4 Why a Process Algebra for GSMCs?

In this section, we motivate the need for a process algebraic language for the specification of GSMCs by discussing a couple of examples. For each example a short informal description is given, a GSMC, and a process algebraic specification. Although using the above description the dynamics of our example GSMCs can be determined, it is in absence of any further explanation not easy to understand. As we will illustrate, this is basically due to the fact that the individual system components are hard to recognize from the overall system structure. This problem becomes more apparent if we consider GSMCs modeling systems of more realistic magnitude. We say that GSMC specifications lack *compositionality*. The idea that we shall pursue here is to specify GSMCs in a compositional way and to exploit this compositional structure by re-using components. We start with a process algebra specification for the G/G/1 queueing system of Example 1.

4.1 The Simple Queueing System

In process algebra, the specification of our queueing system can be obtained in a hierarchical manner, starting from the specifications of the individual components: the queue, the server, and the arrival process. The buffer of infinite capacity can be specified by the set of processes:

$$\begin{aligned} Queue_0 &= a; Queue_1 \\ Queue_i &= a; Queue_{i+1} + b; Queue_{i-1} \text{ for } i > 0 \end{aligned}$$

where the process indices indicate the number of jobs in the buffer, action a denotes enqueueing a job, and action b denotes dequeuing a job. Similarly to GSMCs, clocks can be used to model probabilistic delays. We obtain for the arrival and server processes:

$$\begin{aligned} Arrival &= \mathsf{set}\ x_F\ \mathsf{in}\ (\mathsf{when}\ x_F \mapsto a; Arrival) \\ Server &= b; \mathsf{set}\ y_H\ \mathsf{in}\ (\mathsf{when}\ y_H \mapsto c; Server) \end{aligned}$$

In the *Arrival* process clock x is initialized and starts counting down. Once it has reached the value 0, it expires and action a is enabled. The overall system is described by:

$$System_{G/G/1/\infty} = (Arrival \,||_{\varnothing}\, Server) \,||_{\{a,b\}} Queue_0$$

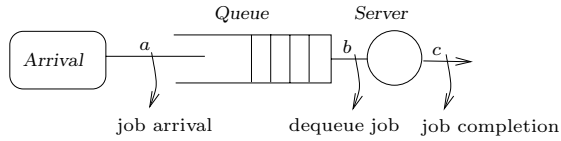


Fig. 5. Compositional specification of G/G/1 queueing system

In process $p \,||_A\, q$, where A is a set of actions, p and q perform actions autonomously, but actions in A should be synchronously performed by both. Action a, for instance, can only take place when the processes *Arrival* and *Queue* are both ready to participate; note that *Server* does not need to participate in this action. The resulting specification of the $G/G/1/\infty$ system closely resembles the structure of the system itself (cf. Fig. 5), it is easy to understand, and it is readily modifiable. For instance, a system with two servers is obtained in the following way:

$$System_{G/G/2/\infty} = (Arrival \,||_\varnothing\, Server \,||_\varnothing\, Server) \,||_{\{a,b\}}\, Queue_0$$

As an alternative extension, a specification of the $G/G/1/K$ queueing system, where K ($K > 0$) denotes the finite capacity of the queue, is obtained by replacing the $Queue_0$ process by a slightly modified buffer $FQueue_0$:

$$\begin{aligned} FQueue_0 &= a; FQueue_1 \\ FQueue_i &= a; FQueue_{i+1} + b; FQueue_{i-1} \text{ for } 0 < i < K \\ FQueue_K &= a; FQueue_K + b; FQueue_{K-1} \end{aligned}$$

where it is assumed that when the queue is full, a job arrival is neglected and lost. The loss of a job is reflected by the fact that for process $FQueue_K$ the capacity is unchanged on arrival of a job.

4.2 A Queueing System with Two Classes of Jobs

Informal description. Consider now a single-server queueing system with a finite queueing capacity $K > 0$, cf. Fig. 6. Two types of jobs are considered. They arrive independently according to different arrival processes. Jobs of class i ($i = 0, 1$) arrive with an inter-arrival time F_i; the processing delay of a job of class i is controlled by distribution H_i. We assume that the service times are mutually independent and are independent of the arrival process. The single server thus takes the job (if any) at the head of the queue and services it with a delay according to its class. The serving discipline is FCFS. (This example is adopted from [21].)

A GSMC description. The possible events in this system are the arrival or completion of a job of class i ($i = 0, 1$); accordingly, the set of events equals $\{ a_0, a_1, c_0, c_1 \}$ where the index of the actions indicate the class they are related to. The description of the state of the system is a bit more involved than for

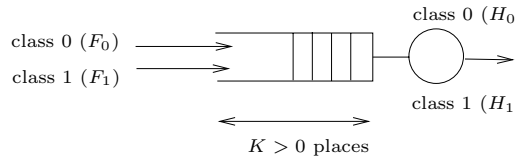


Fig. 6. G/G/1-queueing system with two job classes

the G/G/1/∞ queueing system. It no longer suffices to consider only the queue length, but we also have to keep track of the class of the j-th job in the queue. Thus, a state in the GSMC is a K-tuple $(j_1, \ldots, j_K)$ where $j_m = -$ if the m-th position in the queue is empty, and $j_m = 0$ (1) if this position contains a job of class 0 (1), for $0 < m \leqslant K$. The oldest job (if any) is kept at position 1. This gives rise to $2^{K+1} - 1$ states. The GSMC for K equal to 3 is depicted in Fig. 7 where empty positions in the queue are indicated as blanks. Note that states at depth i indicate scenarios where i jobs are currently in the system.

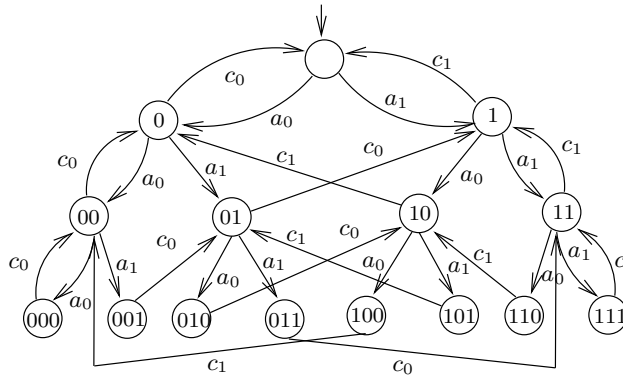


Fig. 7. GSMC of a G/G/1-queueing system with two job classes

A compositional approach. In order to obtain a process algebraic specification of this queueing system we adapt the specification of the simple G/G/1/∞-system in a component-wise manner. The arrival process for each class of jobs is simply an instantiation of the *Arrival* process before; the server is slightly adapted in order to be able to deal with both classes of jobs:

$$\begin{aligned} \mathit{Arrival}_i &= \mathsf{set}\ x_{F_i}\ \mathsf{in}\ (\mathsf{when}\ x_{F_i} \mapsto a_i; \mathit{Arrival}_i) \text{ for } i = 0, 1 \\ \mathit{Server2} &= \quad b_0; \mathsf{set}\ y_{H_0}\ \mathsf{in}\ (\mathsf{when}\ y_{H_0} \mapsto c_0; \mathit{Server2}) \\ &\quad + b_1; \mathsf{set}\ y_{H_1}\ \mathsf{in}\ (\mathsf{when}\ y_{H_1} \mapsto c_1; \mathit{Server2}) \end{aligned}$$

For the finite queue, we extend the specification of the finite queue given above such that, besides the current occupancy of the queue, we keep track of the class

of the j-th job in the queue. The latter is carried out by adding a bit-vector (as superscript) to the process *FQueue* in the following way:

$$\begin{aligned} OQueue_0 &= a_0; OQueue_1^0 + a_1; OQueue_1^1 \\ OQueue_i^{0w} &= a_0; OQueue_{i+1}^{0w0} + a_1; OQueue_{i+1}^{0w1} + b_0; OQueue_{i-1}^{w} \text{ for } 0 < i < K \\ OQueue_K^{0w} &= a_0; OQueue_K^{0w} + a_1; OQueue_K^{0w} + b_0; OQueue_{K-1}^{w} \end{aligned}$$

The processes $OQueue_i^{1w}$ are similar to $OQueue_i^{0w}$ $(0 < i \leqslant K)$ and are omitted here. The overall system is now described by:

$$System_K = (Arrival_0 \,||_\varnothing\, Arrival_1 \,||_\varnothing\, Server2) \,||_{\{a_0,a_1,b_0,b_1\}} OQueue_0 \qquad (5)$$

Note the resemblance with the structure of the G/G/1/∞-specification.

4.3 A Simple Queueing Network

Suppose now that we combine the two queueing systems above in the following (open) queueing network, cf. Fig. 8. The arrival processes of the two classes of jobs in the latter system, $System_K$, are the outputs generated by two finite G/G/1 queueing systems of size K (for class 0) and N (for class 1), respectively. For obtaining the GSMC of this system, the GSMCs of the individual compo-

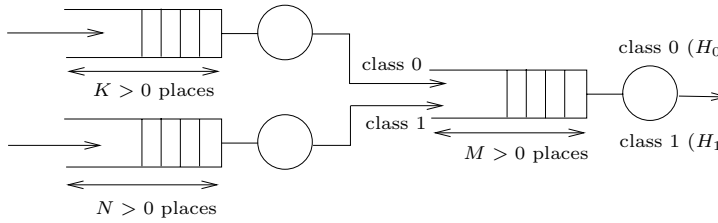


Fig. 8. A simple open queueing network

nents need to be combined in an appropriate way. This is not a straightforward exercise.

To obtain a specification of this system in the process algebraic setting, we first observe that the structure of the queueing network resembles that of $System_K$, except that the input streams are the output streams of two finite G/G/1 systems. Thus, the two *Arrival* processes in specification (5) are replaced by the specifications of the G/G/1-queues. It now remains to "link" the output of the G/G/1 systems to the input of the 2-class buffer system. This is established by means of renaming. Let f be a function that maps action names to action names. Process $p[f]$ behaves like process p except that actions are renamed according to f. For instance, process $System_{G/G/1/K}[c/a_0]$ denotes the finite G/G/1 queue with K places where action c (output) is renamed into a_0.

Thus, a job completion in this system is turned into an arrival of a class 0 job. Using this renaming operator we obtain:

$$\Big(System_{G/G/1/K}[c/a_0] \,||_\varnothing\, System_{G/G/1/N}[c/a_1] \,||_\varnothing\, Server2\Big) \;||_A OQueue_0$$

with $A = \{a_0, a_1, b_0, b_1\}$.

4.4 Non-determinism

Recall that in each state of a GSMC, the next event is determined in a unique way. We like to point out that in the process algebra this is (deliberately) not the case. For instance in a specification like

$$\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; q + \mathsf{when}\ x \mapsto a; r)$$

it is not uniquely determined whether to move to q or to r once clock x has expired. A similar scenario appears in $System_{G/G/2/\infty}$: if the buffer contains a job and both servers are idle, it is non-deterministically determined which server dequeues the job. This phenomenon, known as non-determinism, appears if two (or more) equally labelled transitions become enabled simultaneously. This concept is usually absent in stochastic discrete-event systems – how to analyze their performance? – but has been widely accepted in the computer science community for the purpose of under-specification. This is useful for modeling, for instance [58]:

- Implementation freedom: non-determinism allows to specify freedom of implementation; for instance, if two possible alternatives are described in the specification, a valid implementation would just realize one of them
- Scheduling freedom: if several processes run in parallel, there is a freedom of choice in selecting the next process that performs a move (interleaving)
- External environment: actions represent interaction opportunities with the context in which the process is considered; the interaction capabilities of the environment then influence how the choice is determined

Non-determinism is useful for under-specifying "how often" an alternative is chosen. This information is usually not available in the early steps of the design, or is deliberately left unspecified. If we are to study the performance of such system specifications, this non-determinism will be resolved by adversaries, see Section 7.

5 Stochastic Automata

This section introduces *stochastic automata*, a mixture of timed automata [2] and GSMCs. Stochastic automata are strongly related to GSMCs and incorporate, apart from the necessary (slightly generalized) ingredients to model GSMCs, the possibility of specifying non-determinism. An extensive treatment of stochastic automata is given in [27].

Definition 2. *A* stochastic automaton $SA = (\mathcal{S}, s_0, \mathbf{A}, \mathcal{C}, \longrightarrow, \kappa)$ *where*

- $\mathcal{S}$ *is a non-empty set of* locations *with initial location* $s_0 \in \mathcal{S}$
- $\mathbf{A}$ *is a set of* actions
- $\mathcal{C}$ *is a set of* clocks *with distribution function* F_x *for each* $x \in \mathcal{C}$
- $\longrightarrow \subseteq \mathcal{S} \times (\mathbf{A} \times \mathcal{P}_f(\mathcal{C})) \times \mathcal{S}$ *is a set of* edges
- $\kappa : \mathcal{S} \to \mathcal{P}_f(\mathcal{C})$ *is a* clock-setting function

$(s, a, C, s') \in \longrightarrow$ is denoted $s \xrightarrow{a,C} s'$. To each location s a finite set of clocks $\kappa(s)$ is associated. As soon as location s is entered, any clock x in $\kappa(s)$ is initialized according to its probability distribution function F_x. Once initialized, the clock variables count down at the same rate of letting time pass (like in a GSMC). A clock expires if it has reached the value 0. The occurrence of an action is controlled by the expiration of clocks. Thus, whenever $s \xrightarrow{a,C} s'$ and the system is in location s, action a is offered as soon as all clocks in the set C have expired. In this situation, the edge $s \xrightarrow{a,C} s'$ is called enabled. After taking the edge, the system evolves to location s'. If, after the expiration of a (possibly empty) set of clocks, more than one edge outgoing from s is enabled, an enabled edge is selected non-deterministically.

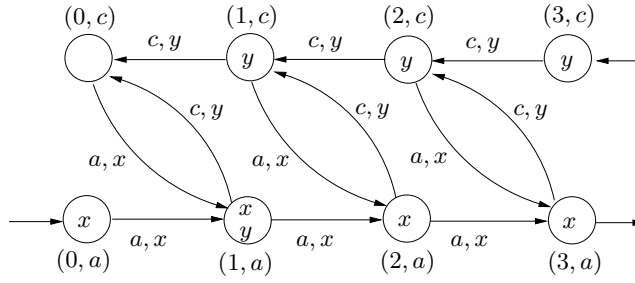


Fig. 9. Stochastic automaton of a $G/G/1/\infty$-system

Example 3. The stochastic automaton $SA_{G/G/1/\infty}$ (cf. Fig. 9) describes the behavior of the $G/G/1/\infty$ queue. Here, we represent a location s as a circle containing the clocks that are to be set in s, and denote edges by arrows. The initial location is represented by a circle equipped with a small ingoing arrow (leftmost circle in second row). We often omit curly brackets for singleton sets. $SA_{G/G/1/\infty}$ is defined by: $\mathcal{S} = \{(i, \alpha) \mid i \geqslant 0, \alpha \in \{a, c\}\}$, $s_0 = (0, a)$, $\mathbf{A} = \{ a, c \}$, $\mathcal{C} = \{ x, y \}$ with $F_x = F$ and $F_y = H$, and

$$\kappa(i, a) = \begin{cases} \{ x \} & \text{if } i \neq 1 \\ \{ x, y \} & \text{if } i = 1 \end{cases} \quad \text{and} \quad \kappa(i, c) = \begin{cases} \{ x \} & \text{if } i \neq 0 \\ \varnothing & \text{if } i = 0 \end{cases}$$

and $(i, \alpha) \xrightarrow{a,x} (i{+}1, a)$ and $(i{+}1, \alpha) \xrightarrow{c,x} (i, c)$ for $i \geqslant 0$ and $\alpha \in \mathbf{A}$.

This stochastic automaton can be understood as follows. Locations are pairs where the first component indicates the number of jobs in the system (i.e., queue and server), and the second component indicates whether the last action was an arrival (action a) or a completion of a job (action c). Note that after the occurrence of action a a location is reached in which clock x is set. Similarly, after the occurrence of action c, clock y is set (except for location $(0, c)$). Clock x thus controls the job inter-arrival time while clock y controls the service delay. In location $(1, a)$, the job that has just arrived is to be served. Thus both the time of the next job arrival and the time until the job is serviced are determined. Accordingly, clocks x and y are set in location $(1, a)$. In location $(0, c)$, however, the last job has just been served and the delay until the next job arrival has decided before. Accordingly, no clock is set in location $(0, c)$.

5.1 Probabilistic Transition Systems

The formal interpretation of stochastic automata is defined in terms of probabilistic transition systems. Probabilistic transition systems are labelled transition systems that contain two disjoint sets of states: probabilistic and non-deterministic states. A non-deterministic state has zero or more outgoing transitions. These transitions determine how the system evolves from the state to a possibly non-deterministically determined next probabilistic state. A probabilistic state has a single outgoing probabilistic transition. A probabilistic transition is a function that maps a probabilistic state onto a probability space whose sample space ranges over the set of non-deterministic states. Paths through a probabilistic transition system are thus sequences of alternating non-deterministic and probabilistic states.

Definition 3. *Let* $Prob(H)$ *denote the set of probability spaces* $(\Omega, \mathcal{F}, \Pr)$ *such that* $\Omega \subseteq H$.[3] *A* probabilistic transition system *(PTS, for short) is a structure* $PTS = (N, P, \mathcal{L}, T, \longrightarrow, \sigma_0)$ *where*

- N *is a set of* non-deterministic *states*
- P *is a set of* probabilistic *states with* $N \cap P = \varnothing$
- $\mathcal{L}$ *is a set of* labels
- $T : P \to Prob(N)$ *is a (total) function called* probabilistic transition relation
- $\longrightarrow \subseteq N \times \mathcal{L} \times P$ *is the* labelled (or non-deterministic) transition relation
- $\sigma_0 \in P$ *is the* initial *(probabilistic) state*

$(\sigma', \ell, \sigma) \in \longrightarrow$ will be denoted by $\sigma' \xrightarrow{\ell} \sigma$. In the setting of this paper, the set of labels is $\mathcal{L} = \mathbf{A} \times \mathbb{R}_{\geqslant 0}$, where $\mathbf{A}$ is a set of action names. The reals denote the (relative) time at which an action takes place. Transition $\sigma \xrightarrow{a,d} \sigma'$ denotes that the system evolves from non-deterministic state σ to probabilistic state σ' by offering action a after idling precisely d time in state σ.

[3] A probability space $(\Omega, \mathcal{F}, \Pr)$ consists of a (sample) set Ω, a σ-algebra $\mathcal{F}$ and a probability measure $\Pr : \mathcal{F} \to \mathbb{R}_{\geqslant 0}$. More details can be found in [70].

Example 4. Consider an automatic switch that controls a light. Assume that the delay between two successive on-button switchings is governed by a negative exponential distribution with mean 30 minutes and that the switch automatically switches off the light in case the on-button has not been switched for exactly 2 minutes. The behavior of the switch is modeled by the following PTS:

- $N = \mathbb{R}^2 \cup \mathbb{R}$
- $P = \{\sigma_{init}, \sigma_{on}\} \cup (\{\sigma_{off}\} \times \mathbb{R})$
- $\mathcal{L} = \{on, off\} \times \mathbb{R}_{\geqslant 0}$
- $T(\sigma_{init}) = \mathcal{R}(F_{e,30})$, $T(\sigma_{on}) = \mathcal{R}(F_{e,30}, D_2)$, and $T(\sigma_{off}, d) = Triv(d)$, and
- $\longrightarrow$ is the smallest relation such that:
  - $(d, d') \xrightarrow{on(d)} \sigma_{on} \Leftrightarrow 0 \leqslant d \leqslant d'$
  - $(d, d') \xrightarrow{off(d')} (\sigma_{off}, d - d') \Leftrightarrow 0 \leqslant d' \leqslant d$
  - $d \xrightarrow{on(d)} \sigma_{on} \Leftrightarrow 0 \leqslant d$
- $\sigma_0 = \sigma_{init}$

where $\mathcal{R}(F_{e,30})$ is the probability space on the real line with the unique probability measure obtained from $F_{e,30}(t) = 1 - e^{-\frac{t}{30}}$, $\mathcal{R}(F_{e,30}, D_2)$ is the probability space on the real plane obtained from $F_{e,30}$ and $D_2(t) = 0$ for $t < 2$ and 1 otherwise, and $Triv(d)$ is the trivial probability space on $\{d\}$.

The PTS is explained as follows. The system starts in σ_{init}, the state in which the light is off. By taking a probabilistic transition, the time d until switching the light on is determined according to distribution $F_{e,30}$ and the system evolves to state d, where the switch waits until it is switched on. If the light is switched on, the system moves to the probabilistic state σ_{on}. On taking the probabilistic transition from σ_{on} two time instants are randomly determined: time d until the light is switched on (again) and time d' until the light turns itself off. The next non-deterministic state is thus (d, d'). Note that $d' = 2$ with probability one. Now two scenarios may occur. If the light is switched on first (i.e., $d \leqslant d'$), the on-button is switched: $(d, d') \longrightarrow \sigma_{on}$ labelled with action $on(d)$. In this state, both time values will be determined again. In the other case, the light turns off while reaching state $(\sigma_{off}, d - d')$ via performing action $off(d')$, where $d - d'$ is the remaining time until the next switching. From state (σ_{off}, d) the non-deterministic state d is reached with probability one, where the switch waits until it is switched on.

5.2 Interpretation of Stochastic Automata in Terms of PTSs

The formal semantics of a stochastic automaton is defined in terms of a PTS. The relation between stochastic automata and PTSs is similar to the relation between timed automata and timed transition systems. We present the mapping of stochastic automata onto PTSs in an informal way; a formal treatment can be found in [27,31]. The states of the PTS keep track of the current location and the values of all clocks involved. Values of clocks are determined by a *valuation*, a function that assigns to each clock $x \in \mathcal{C}$ its real value $v(x) \in \mathbb{R}$. (We let

$\mathcal{V}$ denote the set of valuations.) States are thus pairs (s, v). To distinguish non-deterministic and probabilistic states we write $[s, v]$ for a non-deterministic state, and use (s, v) for probabilistic states. As above, the labelled transition is labelled with pairs (a, d), for action a and delay d.

Performing an edge in the stochastic automaton is represented by a sequence of two steps. Suppose there is an edge from location s to s' labelled with a, C, and assume the current state is $[s, v]$. If after delaying for some time d, say, all clocks in C have expired, then $[s, v] \xrightarrow{a,d} (s', v')$ with s' the location just reached in the stochastic automaton and v' such that for all clocks d time units have passed: $v'(x) = v(x) - d$. While entering location s' though, the clocks in $\kappa(s')$ need to be set. This is accomplished by a subsequent probabilistic transition starting from (s', v'), the state just reached. For the sake of simplicity, assume that $\kappa(s') = \{ x, y \}$. Then, $T(s', v')$ is a unique probability space induced by the distributions of the clocks, F_x and F_y, in a Borel space on a two-dimensional real hyper-space. The clocks that are not in $\kappa(s')$ keep their value while the clocks x and y are initialized according to their distributions. Thus, in the resulting state $[s', v'']$ the system is still in location s' (as expected), and $v''(z) = v'(z)$ for each z different from x and y, while $v''(x) = F_x()$ and $v''(y) = F_y()$. Note the resemblance of this recipe with step 4. of the procedure in Section 3 that described the dynamics of a GSMC.

There is a subtlety though in the first step of the recipe. According to the above procedure, $[s, v] \rightarrow (s', v')$ if all clocks in C have expired. However, we did not make clear yet whether to take such transition *as soon as* all clocks in C have expired, or whether it is allowed to take it at any time point once they have expired. In the first scenario, no delay is allowed once the clocks have expired – it adheres to the so-called "maximal progress" philosophy – while in the second such delay is allowed. Both interpretations have their own use:

- the notion of maximal progress is appropriate when the stochastic automaton is a *closed* system, i.e., a system which is complete by itself and which needs no further synchronization with other automata. As no further synchronizations are envisaged, there is no need to delay transitions any further once they are enabled, since there will be no further (external) processes that can delay their execution. Such perspective is useful, for instance, to determine the model's performance characteristics.
- to study reachability properties like freedom from deadlock, it is important to observe how the system behaves in an arbitrary context. That is, the interaction of a system with a certain "well-behaved" component may not induce a deadlock, while a "badly-behaved" component could take the system through an undesired path that will end in a deadlock situation. In this *open* system perspective, an action that is enabled may not be executed until the environment is also ready to execute such an action. Therefore, it may not take place as soon as it is enabled.

In the sequel, let $\mathcal{O}[\![SA]\!]^v$ denote the open interpretation of SA with initial valuation v, and $\mathcal{C}[\![SA]\!]^v$ denote the closed interpretation of SA. Note that the

only difference between the closed and open interpretation is how to treat the expiration of clocks: either a delay is allowed once they expire (open), or it is not (closed). All other components of the PTSs $\mathcal{O}[\![SA]\!]$ and $\mathcal{C}[\![SA]\!]$ are the same.

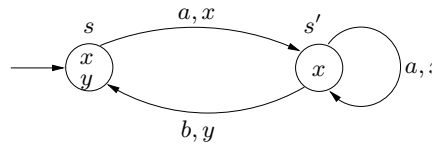


Fig. 10. A simple stochastic automaton

Example 5. To illustrate the difference between the open and closed interpretation, consider the stochastic automaton depicted in Fig. 10. In the closed interpretation the following non-deterministic transitions are present:

$$\begin{array}{ll}
[s,(d,d')] \xrightarrow{a,d} (s',(0,d'-d)) & \text{if and only if } \ d \geqslant 0 \\
[s',(d,d')] \xrightarrow{b,d'} (s,(d-d',0)) & \text{if and only if } \ 0 \leqslant d' \leqslant d \\
[s',(d,d')] \xrightarrow{a,d} (s,(0,d'-d)) & \text{if and only if } \ 0 \leqslant d \leqslant d'
\end{array}$$

where (d,d') denotes valuation v with $v(x) = d$ and $v(y) = d'$. Note the relationship between the clock values of x and y for taking an edge outgoing from location s'. In fact, if F_x and F_y would, for instance, be uniformly distributed on the intervals $[0,5]$ and $[11,12]$, respectively, it follows that in the closed interpretation the edge leading from s' to s will never be enabled. This follows from the fact that there is no valuation (d,d') in s' such that $d' \leqslant d$. Instead on entering location s', clock x is reset and will always expire before clock y expires. In the open interpretation we obtain:

$$\begin{array}{ll}
[s,(d,d')] \xrightarrow{a,d^*} (s',(d-d^*,d'-d^*)) & \text{if and only if } \ d^* \geqslant 0 \wedge d^* \geqslant d \\
[s',(d,d')] \xrightarrow{b,d^*} (s,(d-d^*,d'-d^*)) & \text{if and only if } \ d^* \geqslant 0 \wedge d^* \geqslant d' \\
[s',(d,d')] \xrightarrow{a,d^*} (s,(d-d^*,d'-d^*)) & \text{if and only if } \ d^* \geqslant 0 \wedge d^* \geqslant d
\end{array}$$

In the open interpretation there is no relationship between clock x and clock y. Instead, the only requirement is that the time d^* of taking the transition is positive and beyond the time at which the edge becomes enabled. It follows that in this interpretation the edge from s' to s can indeed be taken for the uniform distributions given above.

5.3 How to Compare Stochastic Automata?

A key question in formal methods (and in practice) is whether a system implementation meets its specification, i.e., when is an implementation proper? In our setting this question can be dealt with in the following way: first model

the behavior of the specification as a stochastic automaton, do the same for the implementation, and then compare these two stochastic automata. Depending on the notion of "being a proper implementation", different comparisons can be made. For instance, if performance is not of much interest one may compare solely on the structure of the two automata while neglecting the probabilistic information. If, on the other hand, performance is of importance, such comparison is insufficient, and probabilistic information should be taken into account. Thus, for different perspectives, different notions of comparison are of interest. As a result, several equivalence relations (i.e., notions of comparison), are defined on stochastic automata.

Isomorphism. The first equivalence notions that we consider is isomorphism. Two stochastic automata are isomorphic if there exists a bijective function that maps locations from one to locations of the other without disturbing the structure of the automaton.

Definition 4. *Stochastic automata* $SA_1 = (\mathcal{S}_1, s_0^1, \mathcal{C}, \mathbf{A}, \longrightarrow_1, \kappa_1)$ *and* $SA_2 = (\mathcal{S}_2, s_0^2, \mathcal{C}, \mathbf{A}, \longrightarrow_2, \kappa_2)$ *are* isomorphic*, denoted* $SA_1 \sim_{iso} SA_2$*, iff there exists a bijection* $\phi : \mathcal{S}_1 \to \mathcal{S}_2$ *such that*

- $\phi(s_0^1) = s_0^2$*, and*
- $s \longrightarrow_1 s'$ *if and only if* $\phi(s) \longrightarrow_2 \phi(s')$*, and*
- $\kappa_1(s) = \kappa_2(\phi(s))$

Note: for simplicity we have assumed that the clocks and actions of both automata are identical; it is not difficult to extend this notion with an isomorphism on clocks and actions.

Structural bisimulation. In concurrency theory, one of the most interesting equivalence relations for labelled transition systems is *bisimulation* [77]: two states are bisimilar if they can mimic each other while evolving into bisimilar states. Two labelled transition systems are bisimilar if and only if their initial states are bisimilar. The notion of structural bisimulation decides the equivalence of stochastic automata by inspecting their structure only.

Definition 5. *Let* $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa)$ *be a stochastic automaton.* $R \subseteq \mathcal{S} \times \mathcal{S}$ *is a* structural bisimulation *if* R *is symmetric and whenever* $s_1 R s_2$*:*

1. $\forall a \in \mathbf{A}, C \subseteq \mathcal{C}.\ s_1 \xrightarrow{a,C} s_1'$ *implies* $\exists s_2'.\ s_2 \xrightarrow{a,C} s_2'$ *and* $s_1' R s_2'$
2. $\kappa(s_1) = \kappa(s_2)$

If R *is a structural bisimulation such that* $s_1 R s_2$*, we write* $s_1 \sim s_2$ *and call* s_1 *and* s_2 *structural bisimilar.*

Stochastic automata SA_1 and SA_2 are *structural bisimilar*, notation $SA_1 \sim SA_2$, if their respective initial locations are structural bisimilar on the disjoint union of SA_1 and SA_2. If we omit the clock-related information, structural bisimulation reduces to usual (strong) bisimulation on labelled transition systems [77].

Example 6. Isomorphic stochastic automata are structural bisimilar, but the reverse is not true, cf. Fig. 11. In the left-hand process, there is a non-deterministic choice in the initial location to move (while emitting a on expiration of clock x) to an absorbing location, i.e., a location without outgoing transitions. There is no bijection that maps the absorbing locations to the single absorbing location in the right-hand process, and thus, these processes are not isomorphic. The automata are structural bisimilar since the relation $R = \{ (s_1, t_1), (s_2, t_2), (s_3, t_2) \}$ is a structural bisimulation.

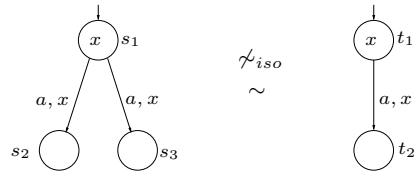


Fig. 11. Two structural bisimilar but non-isomorphic stochastic automata

Probabilistic bisimulation. Structural bisimulation is a simple notion of equivalence and does not take any probabilistic information into account. In order to define a probabilistic variant of bisimulation we consider a notion of *probabilistic bisimulation* on PTSs. Subsequently, we lift this to stochastic automata and illustrate its usage.

Definition 6. *Let $(N, P, \mathcal{L}, T, \longrightarrow)$ be a PTS with $S \subseteq N$ and $\sigma \in P$ such that $T(\sigma) = (\Omega, \mathcal{F}, \Pr)$. Then $\mu : P \times \mathcal{P}(N) \to [0, 1]$ is defined by:*

$$\mu(\sigma, S) = \begin{cases} \Pr(S \cap \Omega) & \text{if } S \cap \Omega \in \mathcal{F} \\ 0 & \text{otherwise} \end{cases}$$

Let R be an equivalence relation on $N \cup P$ such that if $\langle \sigma_1, \sigma_2 \rangle \in R$ then either $\sigma_1, \sigma_2 \in N$ or $\sigma_1, \sigma_2 \in P$. Let N/R be the set of equivalence classes in N induced by R. R is a probabilistic bisimulation *if for all $\langle \sigma_1, \sigma_2 \rangle \in R$:*

1. *$\forall S \subseteq N/R$: $\mu(\sigma_1, \bigcup S) = \mu(\sigma_2, \bigcup S)$, whenever $\sigma_1, \sigma_2 \in P$*
2. *$\forall \ell \in \mathcal{L}$: $\sigma_1 \xrightarrow{\ell} \sigma_1'$ implies $\sigma_2 \xrightarrow{\ell} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in R$, for some $\sigma_2' \in P$, whenever $\sigma_1, \sigma_2 \in N$*

States σ_1 and σ_2 are probabilistically bisimilar*, denoted $\sigma_1 \sim_p \sigma_2$, if there exists a probabilistic bisimulation R with $\langle \sigma_1, \sigma_2 \rangle \in R$. PTS_1 and PTS_2 with initial state σ_1 and σ_2, respectively, are probabilistic bisimilar if $\sigma_1 \sim_p \sigma_2$ in the (disjoint) union of PTS_1 and PTS_2.*

Due to the involvement of continuous distributions, the definition is slightly more involved than existing definitions that consider only discrete distributions [44,71,90]. Nevertheless, both types of probabilistic bisimulation coincide

on the discrete case. A proof of this fact, together with a proof that $\sim_p$ is an equivalence relation, can be found in [27]. Probabilistic bisimulation is lifted to stochastic automata in the following way:

- SA_1 and SA_2 are *open* probabilistic bisimilar, denoted $SA_1 \sim_p^\circ SA_2$ if and only if $\mathcal{O}[\![SA]\!]^{v_0} \sim_p \mathcal{O}[\![SA]\!]^{v_0}$, for any initial valuation v_0;
- SA_1 and SA_2 are *closed* probabilistic bisimilar, denoted $SA_1 \sim_p^\bullet SA_2$ if and only if $\mathcal{C}[\![SA]\!]^{v_0} \sim_p \mathcal{C}[\![SA]\!]^{v_0}$, for any initial valuation v_0.

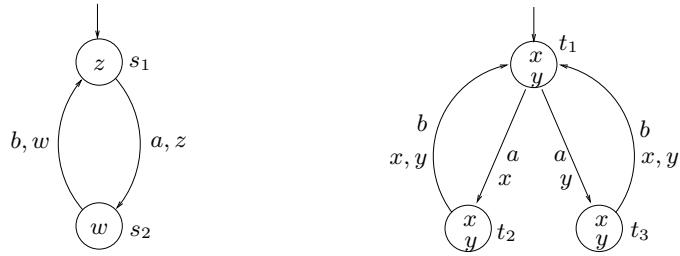


Fig. 12. Two open p-bisimilar stochastic automata

Example 7. The stochastic automata in Fig. 12 are open p-bisimilar if

$$F_z(t) = F_{\min\{x,y\}}(t) = 1 - (1 - F_x(t))(1 - F_y(t)) \text{ and}$$
$$F_w(t) = F_{\max\{x,y\}}(t) = F_x(t) \cdot F_y(t)$$

since

$$\begin{aligned} R = \quad & \{\, ((s_1, v), (t_1, u)) \mid u, v \in \mathcal{V} \,\} \\ & \cup \{\, ([s_1, v], [t_1, u]) \mid u, v \in \mathcal{V}, v(z) = \min\{u(x), u(y)\} \,\} \\ & \cup \{\, ((s_2, v), (t_i, u)) \mid u, v \in \mathcal{V}, i \in \{\, 2, 3 \,\} \,\} \\ & \cup \{\, ([s_2, v], [t_i, u]) \mid u, v \in \mathcal{V}, i \in \{\, 2, 3 \,\}, v(w) = \max\{u(x), u(y)\} \,\} \end{aligned}$$

is a probabilistic bisimulation between their open PTSs. This can intuitively be seen as follows. On entering location t_1 in the right-hand automaton, a race takes place between clocks x and y. According to the clock that expires first — this corresponds to the minimum of their distributions — a transition is made to either t_2 or t_3 while performing a and setting clocks x and y. The same behavior can take place in location s_1 in the left-hand stochastic automaton where a takes place after expiration of clock z with $F_z = F_{\min\{x,y\}}$. From the locations t_2 and t_3 a b-transition is possible back to location t_1 when both clocks x and y — this corresponds to the maximum of their distributions — have expired. This is mimicked by going from s_2 to s_1 after the expiration of clock w with $F_w = F_{\max\{x,y\}}$. Note that it is crucial that both clocks x and y are set in locations t_2 and t_3; otherwise the automata would not have been open probabilistic bisimilar.

Theorem 1. $\sim_p^\bullet \subset \sim_p^\circ \subset \sim \subset \sim_{iso}$.

This result relates the different notions of equivalence that were introduced: isomorphism is the strongest notion, whereas $\sim_p^\bullet$ is the weakest one.

The inclusions in Theorem 1 are strict as exemplified by Fig. 13. Stochastic automata (a) and (b) are structural bisimilar, but not isomorphic, as we have seen before; (b) and (c) are not structural bisimilar, since e.g., the initial locations cannot be related (due to different clocks), but are open probabilistic bisimilar, as both automata can perform an a-action after an equal stochastic delay while evolving to equivalent locations. Finally, the stochastic automata (d) and (e) are closed probabilistic bisimilar as both automata perform an a immediately, but are not open probabilistic bisimilar, because the maximal progress condition does not apply. For instance, a context that is able to participate in b (after imposing a possible extra delay) but forbids action a distinguishes the two processes.

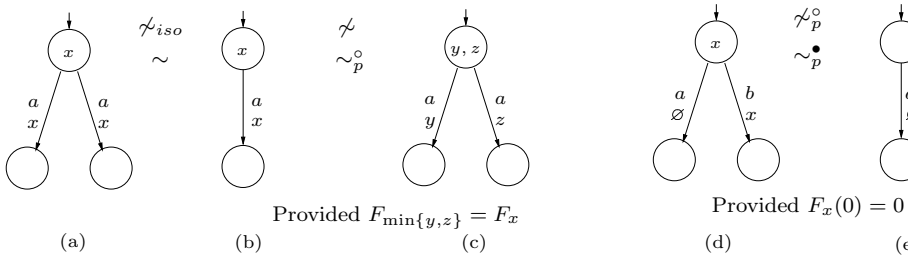


Fig. 13. Structural vs. open vs. closed probabilistic bisimulation

Structural bisimulation is defined directly on stochastic automata, but does not consider any stochastic information. Open and closed probabilistic bisimilarity do take the probabilistic behavior into account, but are defined in terms of the underlying, infinite PTS. Probabilistic information can be considered at a symbolic level too by considering a form of structural bisimulation [27]. The treatment of this notion falls outside the scope of this tutorial.

5.4 GSMCs versus Stochastic Automata

The relation between stochastic automata and GSMCs is shown by providing a mapping, denoted gsmc2sa, from GSMCs onto stochastic automata. The existence of this mapping indicates that GSMCs are properly included in stochastic automata.

The basic idea of the mapping of a GSMC onto a stochastic automaton is to introduce a location as a pair (z, E) where z is a state of the GSMC and E is the set of events that are already active. The initial location is $(z_0, \varnothing)$. For each active event in state z, there is an outgoing edge from any location (z, E). This

edge is labelled with event e (i.e., the action) and the set of clocks $\{C(e)\}$. So, events are considered as actions and active events of z are

$$E(z) = \bigcup \{ e \mid (z, E) \xrightarrow{e,\{C(e)\}} \}.$$

Since in a GSMC exactly one clock is associated to an event, we obtain singleton sets as triggering conditions in the automaton. In the following, we use C on sets of events in the usual way, i.e., $C(E) = \{ C(e) \mid e \in E \}$.

Definition 7. *For GSMC* $\mathcal{G} = (Z, z_0, \mathbf{E}, E, C, \mathsf{next})$ *the stochastic automaton* $\mathsf{gsmc2sa}(\mathcal{G}) = (\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa)$ *is defined by:*

- $\mathcal{S} = Z \times \mathcal{P}_f(\mathbf{E})$ *with* $s_0 = (z_0, \varnothing)$,
- $\mathcal{C} = C(\mathbf{E})$,
- $\mathbf{A} = \mathbf{E}$,
- $\kappa(z, E) = C(E(z) - E)$, *and*
- $\longrightarrow$ *is defined by the rule*

$$\frac{e \in E(z)}{(z, E) \xrightarrow{e,\{C(e)\}} (\mathsf{next}(z, e), E(z) - \{e\})}$$

Due to the fact that $E(z) \neq \varnothing$ for any z, the condition $e \in E(z)$ is always satisfied. There are many locations $(z, E) \in \mathcal{S}$ that are unreachable via $\longrightarrow$. All reachable locations have the form $(\mathsf{next}(z, e), E(z) - \{e\})$ for every (reachable) $z \in Z$ and $e \in E(z)$. Note that for $z' = \mathsf{next}(z, e)$ we have $\kappa(z', E(z) - \{e\}) = C(E(z') - (E(z) - \{e\}))$, the set of clocks for all newly active events in z'.

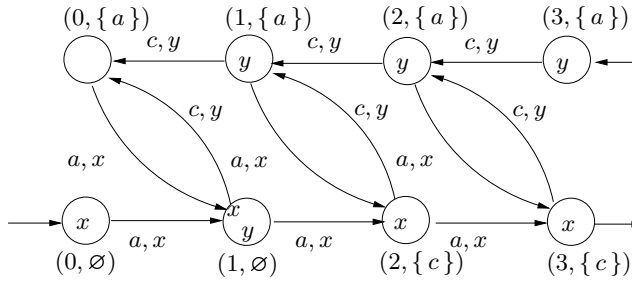


Fig. 14. Stochastic automaton generated from example GSMC of Fig. 3

Example 8. Consider the GSMC of the G/G/1/∞-system depicted in Fig. 3. The stochastic automaton that corresponds to this GSMC is obtained in the following way: the initial location is $(0, \varnothing)$, the state in which there are no jobs in the queue, and there is no active event; $\kappa(0, \varnothing) = C(E(0)) = \{ x \}$; since $a \in E(0)$, and $C(a) = x$, the initial location has a single outgoing edge labelled a, x leading to location $(\mathsf{next}(a, 0), E(0) - \{ a \}) = (1, \varnothing)$ with $\kappa(1, \varnothing) = C(E(1)) = \{ x, y \}$.

It is easy to check that this location has an edge labelled c, y to location $(0, \{ a \})$ and an edge labelled a, x to $(2, \{ c \})$. If we continue this reasoning we obtain the automaton of Fig. 14. Note that this automaton is isomorphic to Fig. 9.

The correctness of the translation gsmc2sa is assessed in [27]. As argued before, stochastic automata are more expressive than GSMCs, since stochastic automata do allow non-determinism (two or more outgoing edges that are enabled at the same time), whereas GSMCs do not. Therefore a reverse translation does not make much sense. In addition, in the stochastic automaton model clocks may be initialized by general distributions — including discrete distribution functions — without any restriction.

6 The Stochastic Process Algebra

The basic idea of the stochastic process algebra SPADES (Stochastic Process Algebra for Discrete-Event Simulation), symbolized by , is to separate the three ingredients that are present in stochastic automata at a syntactical level. We thus distinguish in explicitly between:

- the start of a probabilistic delay (denoted $\mathsf{set}\ C\ \mathsf{in}\ p$),
- the completion of a probabilistic delay (denoted $\mathsf{when}\ C \mapsto p$), and
- the occurrence of immediate actions (denoted $a; p$).

As we will see, this separation allows us to obtain a straightforward expansion law.

6.1 Syntax and Semantics

Syntax. Let $\mathbf{A}$ be a set of *actions*, $\mathbf{V}$ a set of *process variables*, and $\mathcal{C}$ a set of clocks with $(x, G) \in \mathcal{C}$ for x a clock name and G an general probability distribution function. We abbreviate (x, G) by x_G.

Definition 8. *The syntax of is defined by:*

$$p ::= \mathbf{0} \mid a; p \mid \mathsf{when}\ C \mapsto p \mid p + p \mid \mathsf{set}\ C\ \mathsf{in}\ p \mid p \,||_A\, p \mid p[f] \mid X.$$

where $C \subseteq \mathcal{C}$ is finite, $a \in \mathbf{A}$, $A \subseteq \mathbf{A}$, $f : \mathbf{A} \to \mathbf{A}$, and $X \in \mathbf{V}$. A recursive specification *E is a set of recursive equations of the form $X = p$ for each $X \in \mathbf{V}$, where $p \in$.*

Besides the operations used in Section 4 the language incorporates the basic process $\mathbf{0}$, the process that cannot perform any action. A few words on $p + q$ are in order. $p + q$ behaves either as p or q, but not both. At execution the fastest process, i.e. the process that is enabled first, is selected. This is known as the race condition. If this fastest process is not uniquely determined, a non-deterministic selection among the fastest processes is made.

Operators when $C \mapsto \ldots$, set C in $\ldots$, prefixing and renaming have the highest binding precedence, followed by choice and parallel composition.

Semantics. To associate a stochastic automaton $SA(p)$ to a given term p in the language, we define the different components of $SA(p)$[4]. In order to define the automaton associated to a parallel composition, we introduce the additional operation nock. nock(p) is a process that behaves like p except that no clock is set at the very beginning. As usual in structured operational semantics, a location corresponds to a term. The clock setting function κ is defined as the smallest set satisfying the equations in Table 1. The set of edges $\longrightarrow$ between locations is

$\kappa(\mathbf{0})$	$= \varnothing$	$\kappa(\text{set } C \text{ in } p)$	$= \kappa(p) \cup C$
$\kappa(a; p)$	$= \varnothing$	$\kappa(p \,\|\|_A\, q)$	$= \kappa(p) \cup \kappa(q)$
$\kappa(\text{when } C \mapsto p)$	$= \kappa(p)$	$\kappa(p[f])$	$= \kappa(p)$
$\kappa(p + q)$	$= \kappa(p) \cup \kappa(q)$	$\kappa(X)$	$= \kappa(p) \quad (X = p)$
$\kappa(\text{nock}(p))$	$= \varnothing$		

Table 1. Clock setting function for

defined as the smallest relation satisfying the rules in Table 2. The function F is defined by $F(x_G) = G$ for each clock x in p. The other components are defined as for the syntax of .

Let us briefly explain the operational rules from Table 2.

- There is no rule for the process $\mathbf{0}$ as it cannot perform any action.
- The action-prefixed process $a; p$ can immediately perform an a while evolving into p. Since a is performed immediately, there is no need to wait for the expiration of a clock. So, the transition is labelled with an empty set of clocks.
- Process when $C \mapsto p$ can perform any action that p can perform, with the restriction that it has to wait until all clocks in the set C have expired. So, if p has to wait for the expiration of all clocks in C' to perform action a, then process when $C \mapsto p$ has to wait for all clocks in $C \cup C'$.
- Processes set C in p and nock(p) can mimic p; their difference with p is solely in the clock-setting function, cf. Table 1.
- $p + q$ behaves like either p or q (but not both).
- $p[f]$ behaves like p except that all actions are renamed according to f.
- Process X behaves like p, provided it is defined as $X = p$.

[4] Here we assume that p does not contain any name clashes of clock variables. This is not a severe restriction since terms that suffer from such name clash can always be properly renamed into a term without such name clash [27].

$$a;p \xrightarrow{a,\varnothing} p \qquad \frac{p \xrightarrow{a,C'} p'}{\mathsf{when}\ C \mapsto p \xrightarrow{a,C\cup C'} p'} \qquad \frac{p \xrightarrow{a,C'} p'}{\mathsf{set}\ C\ \mathsf{in}\ p \xrightarrow{a,C'} p'}$$

$$\frac{p \xrightarrow{a,C} p'}{\begin{array}{l} p+q \xrightarrow{a,C} p' \\ q+p \xrightarrow{a,C} p' \end{array}} \qquad \frac{p \xrightarrow{a,C} p'}{p[f] \xrightarrow{f(a),C} p'[f]} \qquad \frac{p \xrightarrow{a,C} p'}{\mathsf{nock}(p) \xrightarrow{a,C} p'} \qquad \frac{p \xrightarrow{a,C} p'}{X \xrightarrow{a,C} p'}(X = p)$$

$$\frac{p \xrightarrow{a,C} p'}{\begin{array}{l} p\,||_A\,q \xrightarrow{a,C} p'\,||_A\,\mathsf{nock}(q) \\ q\,||_A\,p \xrightarrow{a,C} \mathsf{nock}(q)\,||_A\,p' \end{array}}(a \notin A) \qquad \frac{p \xrightarrow{a,C} p' \quad q \xrightarrow{a,C'} q'}{p\,||_A\,q \xrightarrow{a,C\cup C'} p'\,||_A\,q'}(a \in A)$$

Table 2. Structured operational semantics for

– For parallel composition two situations are distinguished:
  - In case a synchronization takes place, i.e., some action $a \in A$ is performed, both involved processes must be ready to perform a. So, all clocks needed to perform a in both processes have to be expired.
  - If a process carries out an action not in A, it does so autonomously. Naively, this yields the following traditional operational rule:

$$\frac{p \xrightarrow{a,C} p'}{p\,||_A\,q \xrightarrow{a,C} p'\,||_A\,q}$$

    for $a \notin A$. This would, however, lead to a situation in which all clocks in p' and q are reset in the resulting location. This is incorrect for the clocks in q, since now the elapse of time since the clocks of q were set (when reaching $p\,||_A\,q$) is neglected, cf. equation (4) and Fig. 2. To solve this problem, the state $p'\,||_A\,\mathsf{nock}(q)$ is reached instead where the use of $\mathsf{nock}(q)$ avoids the setting of the clocks in q, i.e., $\kappa(\mathsf{nock}(q)) = \varnothing$, cf. Table 1.

Example 9. Using this recipe it can be shown that the semantics of the process algebraic G/G/1/∞ specification boils down to the (at first sight somewhat complicated) stochastic automaton depicted in Fig. 15. Here, empty sets are omitted; in particular b stands for $b, \varnothing$. Note that the in locations that are reached after the server has just completed servicing a job, no clocks are set. Although the state space of this automaton is somewhat larger than that of the direct representation in Fig. 14, this does not have a serious impact on the efficiency of stochastic simulation, as will see later on. Since in our semantics a state corresponds to a term, simulation can be carried out on the basis of expressions rather than using their semantic representations. This will be further discussed in Section 7.

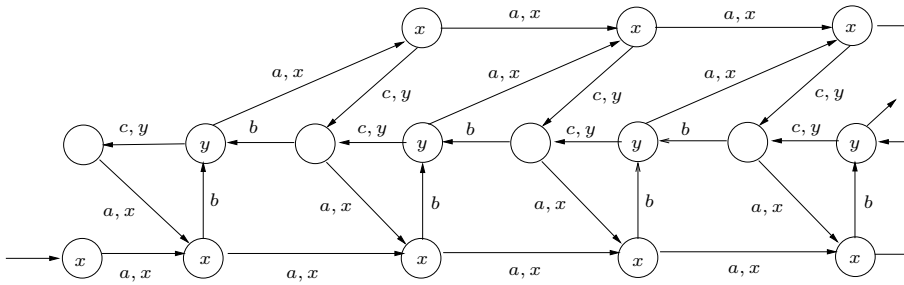


Fig. 15. Stochastic automaton of the compositional $G/G/1/\infty$ specification

Expressive power of stochastic automata versus ⏢. Stochastic automata and ⏢ are equally expressive. This means that for any stochastic automaton a corresponding term in ⏢ can be given whose reachable part of its stochastic automaton is identical to the stochastic automaton at hand, up to renaming of clocks.

Theorem 2. *For every stochastic automaton SA there exists a recursive specification E with root p in* ⏢ *such that the reachable part of SA and the reachable part of* $SA(p)$ *are isomorphic.*

As a result of this theorem and the fact that GSMCs are a proper subset of stochastic automata, it follows that each GSMC is representable by a ⏢ specification.

Example 10. Consider the stochastic automaton of Fig. 9. The reader is invited to check that this stochastic automaton is isomorphic to the following recursive ⏢ specification for $i \geqslant 0$:

$$
\begin{aligned}
P_0 &= \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; P_1) \\
P_1 &= \mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto c; Q_0 + \mathsf{when}\ x \mapsto a; P_2) \\
P_{i+2} &= \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ y \mapsto c; Q_{i+1} + \mathsf{when}\ x \mapsto a; P_{i+3}) \\
Q_0 &= \mathsf{set}\ \varnothing\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; P_1) \\
Q_{i+1} &= \mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; P_{i+2} + \mathsf{when}\ y \mapsto c; Q_i)
\end{aligned}
$$

The indices the processes indicate the number of jobs that are currently in the system, i.e., that are either queued or currently being processed. Note that the structure of each process definition is of the same shape: first some clocks are set, then their expiration is awaited, and an action takes place before invoking a process instance.

6.2 Notions of Congruence

The equivalence notions defined for stochastic automata (cf. Section 5.3) can be lifted to terms in ⏢ in the following straightforward way. Terms p and q are

structurally bisimilar, denoted $p \sim q$, if and only if $SA(p) \sim SA(q)$. In a similar way we define: $p \sim_p^\circ q$ iff $SA(p) \sim_p^\circ SA(q)$ and $p \sim_p^\bullet q$ iff $SA(p) \sim_p^\bullet SA(q)$. Here, we will investigate whether these equivalence notions are congruences. Recall that an equivalence relation is a congruence if two equivalent terms behave indistinguishable in any context. We have:

Theorem 3. *$\sim$ and $\sim_p^\circ$ are congruences with respect to all operators in* $\spadesuit$.

(It should be noted that $\sim$ is also a congruence for recursion and nock.) Due to this result, replacing in a $\spadesuit$ specification a sub-term p by its bisimilar equivalent q results in a bisimilar $\spadesuit$ specification. The proof of this result for $\sim$ is rather simple and follows from the fact that the operational rules of Table 2 obey a certain syntactic format (the so-called path format of [7]); the proof for $\sim_p^\circ$ is rather involved and tedious as it requires manipulations on Borel spaces, cf. [27]. $\sim_p^\bullet$ is not a congruence for parallel composition as illustrated by the following example.

Example 11. Consider the processes

$$p = a;\mathbf{0} + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto\ b;\mathbf{0})$$
$$q = a;\mathbf{0} + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto\ c;\mathbf{0})$$

where b and c are distinct actions. Note that $q = p[b/c]$. We have $p \sim_p^\bullet q$ if $F_x(0) = 0$, since then in both processes only action a at time 0 can be performed due to the maximal progress principle. However,

$$p \,||_a\, \mathbf{0} \not\sim_p^\bullet q \,||_a\, \mathbf{0}$$

In the context process $\mathbf{0}$, the execution of action a is disabled, since there is no possible synchronization, and therefore b or c will happen (at a certain positive time instant). This example is depicted in terms of stochastic automata in Fig. 16.

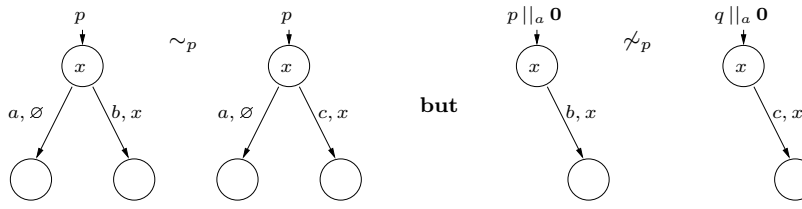


Fig. 16. Closed probabilistic bisimulation is not a congruence for $||_A$

The reader is invited to check that $\sim_p^\bullet$ is also not a congruence for the operator $\mathsf{when}\ C \mapsto p$.

6.3 Equational Reasoning

Rather than proving that p and q are e.g., structural bisimilar using the semantic interpretations $SA(p)$ and $SA(q)$ it is often more convenient to use rules defined on the syntax of p and q that are known to preserve (in this case) $\sim$. This enables the transformation and comparison of terms at a purely syntactic level. In this section, we present a sound and complete axiomatization for structural bisimulation of the core calculus of 🔔, i.e., the fragment of 🔔 consisting of the basic operators, i.e., excluding recursion and parallel composition. In addition, some sound axioms for open probabilistic bisimulation will be presented.

Axiomatization of structural bisimulation. Let the set $fv(p)$ of free (clock) variables of p be defined as the smallest set satisfying the equations in Table 3. The

$fv(\mathbf{0})$	$= \varnothing$	$fv(\mathsf{set}\ C\ \mathsf{in}\ p)$	$= fv(p) - C$
$fv(a;p)$	$= fv(p)$	$fv(p \,\|\|_A\, q)$	$= fv(p) \cup fv(q)$
$fv(\mathsf{when}\ C \mapsto p)$	$= C \cup fv(p)$	$fv(p[f])$	$= fv(p)$
$fv(p+q)$	$= fv(p) \cup fv(q)$	$fv(X)$	$= fv(p)$ $\quad (\mathrm{X} = \mathrm{p})$
$fv(\mathsf{nock}(p))$	$= fv(p) \cup \kappa(p)$		

Table 3. Free variables of terms in 🔔

axioms for structural bisimulation are given in Table 4 and can be explained as follows. As in traditional process algebra, the choice is commutative (**A1**) and associative (**A2**), and **0** is the neutral element for $+$ (**A4**). Axiom **A3** is a distinguishing law for stochastic process algebra (also for Markovian process algebra) and can be regarded as a weak version of the traditional idempotence axiom of choice ($p + p = p$). The reason of not having this axiom of choice is that in case of two competing processes such as

$$(\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p)) + (\mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p))$$

the resolution of the choice is controlled by the minimum of two random variables with the distributions of x and y, respectively. As a result, the term p is reached "faster" than in either of the sub-terms of the choice. Accordingly,

$$\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p) + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p) \neq \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p) \tag{6}$$

Later on, we will make this more precise when discussing some axioms of open probabilistic bisimulation. Note that, however, due to **T5** and **A3** we have:

$$\mathsf{set}\ x\ \mathsf{in}\ ((\mathsf{when}\ x \mapsto a;p) + (\mathsf{when}\ x \mapsto a;p)) = \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p)$$

A1	$p + q = q + p$	
A2	$(p + q) + r = p + (q + r)$	
A3	$a; p + a; p = a; p$	
A4	$p + \mathbf{0} = p$	
T1	$\mathsf{when}\ C \mapsto \mathbf{0} = \mathbf{0}$	
T2	$\mathsf{when}\ \varnothing \mapsto p = p$	
T3	$\mathsf{when}\ C \mapsto (\mathsf{when}\ C' \mapsto p) = \mathsf{when}\ (C \cup C') \mapsto p$	
T4	$\mathsf{when}\ C \mapsto (\mathsf{set}\ C'\ \mathsf{in}\ p) = \mathsf{set}\ C'\ \mathsf{in}\ (\mathsf{when}\ C \mapsto p)$	if $C \cap C' = \varnothing$
T5	$\mathsf{when}\ C \mapsto (p + q) = \mathsf{when}\ C \mapsto p + \mathsf{when}\ C \mapsto q$	
C1	$\mathsf{set}\ \varnothing\ \mathsf{in}\ p = p$	
C2	$\mathsf{set}\ C\ \mathsf{in}\ (\mathsf{set}\ C'\ \mathsf{in}\ p) = \mathsf{set}\ (C \cup C')\ \mathsf{in}\ p$	
C3	$(\mathsf{set}\ C\ \mathsf{in}\ p) + (\mathsf{set}\ C'\ \mathsf{in}\ q) = \mathsf{set}\ (C \cup C')\ \mathsf{in}\ (p + q)$	if $C \cap (fv(q) \cup \kappa(q)) = C' \cap (fv(p) \cup \kappa(p)) = \varnothing$

Table 4. Axioms for structural bisimulation on 

where the setting of clock x is common to both terms. In fact, one can show that the idempotence law $p + p = p$ is obtained if the structure of p is restricted such that all clock setting operations of the form $\mathsf{set}\ C\ \mathsf{in}\ r$ in p occur in a sub-term of the form $a; q$. Axioms **T1**–**T5** show the way in which triggering conditions can be simplified. In particular, **T3** defines how to reduce nested triggering conditions into a single one, and axioms **T4** and **T5** state how to move clock settings and summations out of the scope of a guard. Axiom **C1** expresses that it is irrelevant to set an empty set of clocks. **C2** gathers all the clocks settings in a single operation and **C3** moves clocks settings out of the scope of a summation. (The auxiliary notion of free variables in a term is defined in Table 3.)

For axioms for the static operators such as renaming and parallel composition we refer to [27].

Expansion law. Using the complete and sound axiomatization of structural bisimulation for  one can show that any (finite) term p can be converted into a normal form which has the shape

$$\mathsf{set}\ C\ \mathsf{in}\ \left(\sum \mathsf{when}\ C_i \mapsto a_i; p_i\right)$$

where p_i are terms in normal form and $\sum$ is the usual generalization of choice: $\sum_{0<i\leqslant n} p_i$ equals $p_1 + \ldots + p_n$ for $n{>}0$, and $\mathbf{0}$ for $n{=}0$. The reader is invited to check that this format is the same as the one used in Example 10.

As we have argued in the introduction, an essential law in traditional process algebras is the expansion law. This law allows to reduce parallel composition in terms of prefix and choice, and has proven to be of crucial importance for process

algebraic verification purposes. A stochastic equivalent of this law is defined by the following result.

Theorem 4. *Let* $p, q \in$ *such that* $p = \mathsf{set}\ C\ \mathsf{in}\ p'$ *and* $q = \mathsf{set}\ C'\ \mathsf{in}\ q'$ *with* $p' = \sum(\mathsf{when}\ C_i \mapsto a_i; p_i)$ *and* $q' = \sum(\mathsf{when}\ C'_j \mapsto b_j; q_j)$. *Suppose* $p \,||_A\, q$ *does not contain name clashes. Then* $p \,||_A\, q$ *equals*

$$\begin{aligned}\mathsf{set}\ (C \cup C')\ \mathsf{in}\ \Big(& \textstyle\sum_{a_i \notin A} \mathsf{when}\ C_i \mapsto a_i; (p_i \,||_A\, q') \\ & + \textstyle\sum_{b_j \notin A} \mathsf{when}\ C'_j \mapsto b_j; (p' \,||_A\, q_j) \\ & + \textstyle\sum_{a_i = b_j \in A} \mathsf{when}\ (C_i \cup C'_j) \mapsto a_i; (p_i \,||_A\, q_j)\ \Big).\end{aligned}$$

In fact, the expansion law is inherent in our model and follows at the language level from the way in which we have distinguished between 3 activities in the syntax: (1) starting a delay, i.e., setting a clock, (2) finishing a delay, i.e., expiration of a clock, and (3) the occurrence of (immediate) actions.

Example 12. By means of the above expansion law we would like to discuss the way in which delayed actions are synchronized in  . For that purpose we consider:

$$(\mathsf{set}\ x\ \mathsf{in}\ \mathsf{when}\ x \mapsto a; p) \,||_a\, (\mathsf{set}\ y\ \mathsf{in}\ \mathsf{when}\ y \mapsto a; q)$$

Using the expansion law, this term can be rewritten into the equivalent term

$$\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x, y \mapsto a; (p \,||_a\, q))$$

This entails that action a can happen after both clocks x and y have expired. In stochastic terms, this means that two random variables are competing, viz. the maximum of the random variables with distributions F_x and F_y. As the maximum of two statistically independent random variables is distributed according to the product of their distribution (cf. Example 7), we obtain that synchronization of delayed actions in  is based on the product of distributions. In most Markovian process algebras (with the notable exception of [49,51]), this policy is not taken, since the class of exponential distributions is not closed under product.

Open probabilistic bisimulation. Here, we present some sound axioms for open probabilistic bisimulation ($\sim_p^\circ$). The axioms presented in Table 5 are incomplete, but are useful for reducing the number of clocks in a  expression. Axiom **P1** allows to eliminate redundant clock settings by stating that it is not necessary to set clocks that do not occur free in process p. Since these clocks do not freely occur in p they are either not used or are set again once they will be used. Axiom **P2** states that clocks that have been used, i.e, that have expired, are not useful anymore (at least not before being set again), and thus can safely be removed. Axiom **P3** expresses that clocks that are set to a positive value with probability 0 cannot affect the timing of a process. Finally, the most involved axiom is **A3′**. This axiom is a weak variant of the idempotence axiom **A3** (i.e.,

P1	$\mathsf{set}\ C\ \mathsf{in}\ p = p$	if $C \cap fv(p) = \varnothing$
P2	$(\mathsf{when}\ C \mapsto a); (\mathsf{when}\ C \mapsto p) = \mathsf{when}\ C \mapsto a; p$	
A3′	$\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p + \mathsf{when}\ y \mapsto p) = \mathsf{set}\ z\ \mathsf{in}\ (\mathsf{when}\ z \mapsto p)$ if $\{ x, y, z \} \cap fv(p) = \varnothing \wedge \forall t \in \mathbb{R}_{\geqslant 0}.\, F_z(t) = F_{\min\{x,y\}}(t)$	
P3	$\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) = \mathsf{set}\ x\ \mathsf{in}\ p$	if $F_x(0) = 1$

Table 5. Some axioms for open probabilistic bisimulation on

$a; p + a; p = a; p$) for structural bisimulation. Axiom **A3′** states that in case of two competing processes such as

$$(\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p)) + (\mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p))$$

which, according to **C3**, is equivalent to

$$\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p + \mathsf{when}\ y \mapsto p)$$

the resolution of the choice is controlled by the minimum of two independent random variables with the distributions of x and y. The process can thus be replaced by

$$\mathsf{set}\ z\ \mathsf{in}\ (\mathsf{when}\ z \mapsto p)$$

where z is a fresh clock (i.e., $\{ x, y, z \} \cap fv(p) = \varnothing$) with distribution $min(F_x, F_y)$. Note that in case x and y are exponentially distributed, axiom **A3′** reduces to the idempotence law for Markovian process algebra [52]:

$$\lambda \mapsto p + \mu \mapsto p = (\lambda + \mu) \mapsto p$$

as stated in the notation used in the introduction.

Example 13. Using axiom **A3′** we can now deduce that indeed inequation (6) is valid in general:

$$\begin{aligned}
& \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) \\
= \quad & \{\ \alpha \text{ conversion}\ \} \\
& \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) + \mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p) \\
= \quad & \{\ \text{axiom } \mathbf{C3}\ \} \\
& \mathsf{set}\ x, y\ \mathsf{in}\ ((\mathsf{when}\ x \mapsto p) + (\mathsf{when}\ y \mapsto p)) \\
= \quad & \{\ \text{axiom } \mathbf{A3'}\ \} \\
& \mathsf{set}\ z\ \mathsf{in}\ (\mathsf{when}\ z \mapsto p)
\end{aligned}$$

where $F_z = F_x \cdot F_y = F_x^2$. Since $F_x^2 \neq F_x$ for any non-trivial random variable x, we obtain the inequality (6).

7 Analysis of Specifications

Until so far, we have introduced a process algebra for describing GSMCs (with possible non-determinism) while concentrating on notions like formal semantics, equivalences, axiomatisation and so on. In this section, we focus our attention on the *analysis* of specifications, both in a quantitative and qualitative sense. For the first type of analysis we consider discrete-event simulation, for the qualitative analysis we consider the checking of reachability properties.

7.1 Quantitative Analysis: Discrete-Event Simulation

By means of quantitative analysis, we obtain insight in questions related to the performance (in measures like throughput or response times) and dependability of systems (in terms of failure rates, mean time between failure, and so on). For a restricted set of stochastic automata – those that correspond to insensitive GSMCs [88] – numerical methods can be used to assess their steady-state (i.e., long run) behavior. Alternatively, in case of absence of non-determinism, approximation results can be employed, and arbitrary distributions can be approximated by phase-type distributions [79]. This results in a continuous-time Markov chain for which efficient numerical methods exist [95].

A general approach towards assessing quantitative properties is simulation, in particular *discrete-event simulation* [21,92]. In this simulation technique state changes take place at discrete points in time (but time is continuous), as opposed to continuous-time simulation techniques. In a simulation, runs (sample paths in the simulation jargon) are generated, and on the basis of these runs data is gathered and analyzed to determine an estimate of the desired measure of interest. The accuracy of the estimate is given by a confidence interval.

In the rest of this section, we focus on the following issues that occur when applying discrete-event simulation on specifications:

- how to generate sample paths from stochastic automata, and
- how to resolve non-determinism in a stochastic automaton prior to the simulation.

More details about simulation of specifications can be found in [27,33].

Runs and adversaries. A run of a labelled transition system is simply a walk through the state space by traversing transitions starting from the initial state. This also applies to PTSs except that we focus on traversals that are "probable".

Definition 9. *A* run ρ *of PTS* $(N, P, \sigma_0, \mathcal{L}, T, \longrightarrow)$ *is a (finite or infinite) sequence* $\rho = \sigma_0\sigma'_0\ell_1 \ldots \ell_n\sigma_n\sigma'_n$ *for* $n \in \mathbb{N} \cup \{\infty\}$ *such that, for all* $0 \leqslant i < n$*:*

- σ'_i *is in the support set of the probability measure of* $T(\sigma_i)$[5]
- $\sigma'_i \xrightarrow{\ell_{i+1}} \sigma_{i+1}$*, and*
- *if* ρ *is finite, then* ρ *ends in a non-deterministic state.*

[5] Intuitively, the support set of a probability measure is the smallest closed (measurable) set which has probability one.

Non-determinism is resolved by probabilistic adversaries: if during a run a state has been reached which has several non-deterministic possibilities, such adversary will make the choice in a (discrete) probabilistic way. An adversary is similar to a policy in Markov decision processes [34]. An *adversary* $\mathcal{A}$ on PTS $\mathcal{T}$ is a partial function that maps a finite run ρ of PTS $\mathcal{T}$ to a discrete probability space on the non-deterministic transition relation. The sample space of $\mathcal{A}$ is a non-empty subset of the set of outgoing transitions from the final state of the run ρ. An adversary $\mathcal{A}$ of stochastic automaton SA is a partial function on the runs of its closed semantics $\mathcal{C}[\![SA]\!]^{v_0}$.[6] Note that an adversary selects the next transition on the entire run ρ, i.e., on the entire history of the system. Usually, adversaries are considered that make a selection based on the current state only. For pragmatic reasons (space efficiency) we confine ourselves to such memoryless adversaries.

Example 14. To illustrate the notion of adversary, consider the 🔔-specification of the queueing system with two servers from Section 4 where we assume that all clocks in the system are governed by continuous distribution functions. The overall system specification was given by:

$$System_{G/G/2/\infty} = (Arrival \,||_{\varnothing}\, Server \,||_{\varnothing}\, Server) \,||_{\{a,b\}}\, Queue_0$$

This system contains non-determinism in case the queue contains one or more jobs and both servers are ready to process a job — which server will take the job out of the queue? In case we do not have any preference of one server over the other (e.g., they are both equally fast), an adversary that resolves this non-deterministic situation with equal probability is appropriate. Then $\mathcal{A}$ is defined such that $\mathcal{A}(\rho)$ equals $\frac{1}{2}$ for the first server taking the job out of the queue and $\frac{1}{2}$ for the second server taking the job out of the queue. In case we would prefer one server over the other, a different choice for $\mathcal{A}$ can be made.

Discrete-event simulation. Once we have resolved the present non-determinism (if any), the resulting system is fully probabilistic, that is to say, a stochastic automaton with an adversary that resolves all its non-determinism yields a GSMC with probabilistic branching. Hence, discrete-event simulation of such systems basically takes place according to the operational procedure as described in Section 3 for GSMCs with the minor difference that the trigger event e^* may have several possible next states that are selected (by the adversary) in a probabilistic way. The user should bear in mind that the simulation results that will be obtained must be considered with respect to the specified adversary as different results are obtained (in general) for different adversaries!

An interesting aspect of simulation is that the state space is generated in an "on-the-fly" fashion – the state space is constructed dynamically and thus requires minimal storage as only the current state needs to be stored. This means that we are not forced to construct the entire stochastic automaton prior to the

[6] There are some conditions on the sample space that we omit here for the sake of simplicity; the interested reader may consult [27,33].

simulation. Instead, it suffices to store only the current state and generate new states on a call-by-need basis. As a consequence, simulation is not restricted to finite stochastic automata, but is also applicable to infinite-state stochastic automata. Besides, the generation of a state is very straightforward for a specification as a state uniquely corresponds to a process algebra expression.

Example 15. Consider the compositional -specification of the G/G/1-queueing system. Let the distribution F of clock x be such that $F(0) = 0$. The simulation will start by generating the initial term. Then clock x is initialized and an arrival (action a) is generated once x expires. The next term (i.e., location) is generated, and a new sample for x is taken. Since $F_x(0) = 0$, this sample is positive. As in this location both an immediate action (b) and a non-immediate action (a) are possible, the latter will never be taken in a simulation since it has to be delayed first. Accordingly action b happens and the next term is reached (after initializing clock y). This procedure continues ad infinitum. Note that the stochastic automaton as depicted in Fig. 14 is actually generated location by location, and that in each step only the current location (plus the valuation of all clocks involved) are needed to determine the next possible steps. Due to choices between immediate actions and non-immediate actions, in fact, only the sub-automaton of Fig. 17 is generated step-by-step. The reduction from Fig. 14 to Fig. 17 can also be carried out in an equational way while preserving probabilistic bisimulation [27].

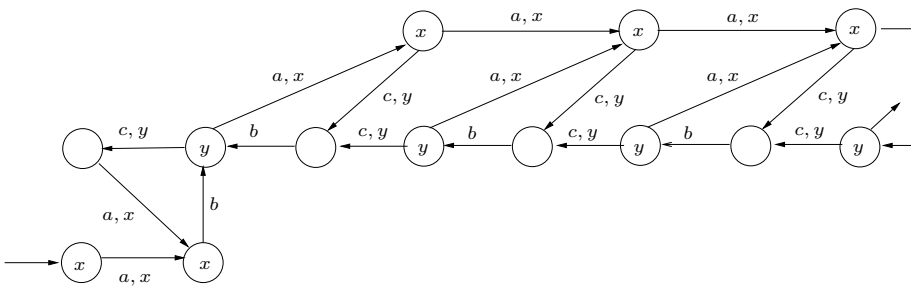


Fig. 17. Reduced stochastic automaton generated by simulation (if $F(0) = 0$)

7.2 Qualitative Analysis: Reachability Properties

Complementary to the quantitative analysis described above, we discuss in this section a classical analysis technique for functional correctness — reachability analysis. Reachability analysis is the key technique in proving safety properties (often characterized as properties of the type "something bad can never happen"). A typical reachability property is the absence of a deadlock, which is a state from which no further progress can be made. A naive strategy would

be to use the simulation approach just described and consider several simulation runs. As the coverage of such simulations is rather low – if no deadlock is reached during simulation, this does not guarantee absence of deadlocks – more systematic approaches are needed. In order to systematically check such properties for stochastic automata, and thus terms, the underlying semantics in terms of PTSs needs to be examined. However, even for finite terms, these transition systems are infinite due to the fact that distributions are continuous. We therefore consider a *symbolic* reachability analysis. Using a symbolic analysis we can avoid having to build and examine the infinite underlying PTS. Instead, we check reachability at the level of stochastic automata. We investigate this for finite stochastic automata.

Reachability. A (non-deterministic) state in a PTS is reachable if there exists a finite run that ends in that state. Let $\mathit{Reach}(\mathcal{T})$ denote the set of reachable states of PTS $\mathcal{T}$.

Definition 10. *Let* $SA = (\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \rightarrow, \kappa, F)$ *be a stochastic automaton. A* symbolic run *of* SA *is a finite sequence* $s_0 a_1 C_1 s_1 \ldots s_{n-1} a_n C_n s_n$, $n \geqslant 0$, *such that, for all* $0 < i \leqslant n$, $s_{i-1} \xrightarrow{a_i, C_i} s_i$.

Location s is reachable if and only if there exists a symbolic run starting from s_0 that ends in s. The set of reachable locations of SA is denoted $\mathit{Reach}(SA)$. The relationship between runs in a stochastic automaton, and the runs in the underlying (open and closed) PTSs is as follows. Every symbolic run of SA has a corresponding finite run in $\mathcal{O}[\![SA]\!]$, and vice versa:

Theorem 5. $s \in \mathit{Reach}(SA) \Leftrightarrow \exists v \in \mathcal{V}.\, [s, v] \in \mathit{Reach}(\mathcal{O}[\![SA]\!]^{v_0})$

Recall that, as opposed to the open interpretation, in the closed interpretation an edge can only be taken, if there is no earlier point in time at which the current location can be left (maximal progress). As a consequence, certain edges present in the stochastic automaton need not result in a transition in the underlying closed PTS, since there exist competitive edges that are "faster" and thus will be taken instead. As a result, every finite run of $\mathcal{C}[\![SA]\!]$ has a corresponding symbolic run of SA,but not the reverse:

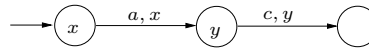
Theorem 6. $s \notin \mathit{Reach}(SA) \Rightarrow \forall v \in \mathcal{V}.\, [s, v] \notin \mathit{Reach}(\mathcal{C}[\![SA]\!]^{v_0})$.

This result is e.g., sufficient to check for freedom of deadlock: if $\mathcal{C}[\![SA]\!]$ does not have a reachable deadlock state, SA is deadlock-free.

These results allow us to carry out systematic reachability analysis at a purely symbolic level, i.e., without the construction of the underlying infinite PTS and without using the clock information in the stochastic automaton. In this way, we can exploit existing tools like SPIN [59] for carrying out the reachability analysis.

Timed reachability properties. For checking reachability of locations, the stochastic information may not be of any importance. However, we like to obtain more information about timing aspects – e.g. "is a certain state reachable within t time units?". In this case, the precise probabilistic value of the occurrence time

of some action is not relevant. Instead, it suffices to consider whether the action is possible or not at a particular time instant. Therefore, only a small bit of the information about the probability distributions involved is necessary. In fact, it suffices to know the "support set". In other words, we only need to consider within which boundaries a delay is possible. For instance, in a simple stochastic automaton like:



where F_x and F_y are uniform distributions on intervals $[10, 20]$ and $[0, 12]$, say, one can infer only from the bounds of the distributions that the right-most location is reachable within 32 time units. This idea is formalized in [28] by a compositional translation from stochastic automata into timed automata with deadlines [14]. This translation preserves safety properties in the sense that any state (i.e. a location and a clock valuation) that is not reachable in the generated timed automaton is also not reachable in the original stochastic automaton. Thus, timed safety properties are preserved.

An overview of the analysis techniques for specifications is given in Fig. 18.

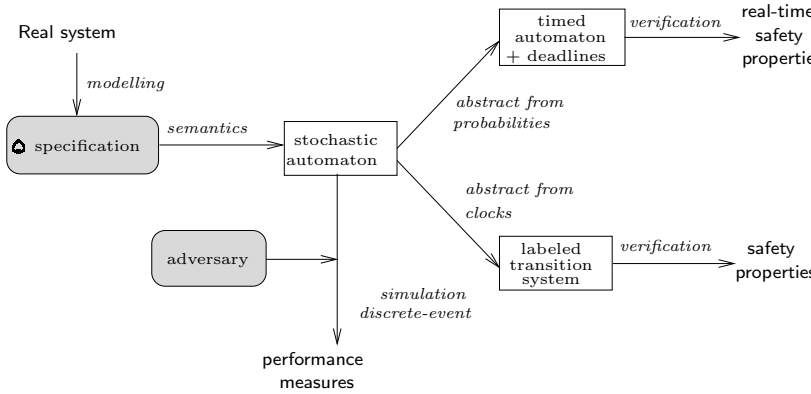


Fig. 18. Analysis of a specification

8 Case Study: The IEEE 1394 Root Contention Protocol

This section discusses a small case study where we applied to specify the root contention phase of the IEEE 1394 protocol in a compositional way, and used discrete-event simulation to analyze the time until contention resolution.

8.1 Informal Problem Description

IEEE 1394. The IEEE 1394 high performance serial bus has been developed for interconnecting computer and consumer equipment such as VCRs, CD players and multimedia PCs. It supports the addition and removal of equipment at any time ("hot-pluggable") and allows quick, reliable and inexpensive high-bandwidth transfer of digitized video and audio. The bus has been originally developed by Apple (FireWire) and has been standardized by IEEE in 1996 [62]. A revision of this standard is currently in progress [63]. As several companies have joined in the development of the 1394 bus, there is a good chance that this will become the future standard for connecting digital multimedia equipment; in fact, the bus is currently also being used in areas like aerospace equipment. Various parts of the standard have been specified and verified formally, e.g. [35,85,96,91]. Here, we consider the so-called root contention protocol.

Leader election protocol. (This description is adopted from [96].) The IEEE 1394 standard consists of a stack of protocols. The physical layer, the lowest in the protocol hierarchy, consists of a number of phases. During the tree identify phase, which is entered on occurrence of a bus reset, it is checked whether the network topology is a tree, and if so, a leader is elected among the nodes in the tree. The leader election takes place while constructing a spanning tree in the network and electing the root of the tree as leader. Informally, the basic idea of the protocol is as follows: leaf nodes send a "parent request" message to their neighbor. When a node has received a parent request from all but one of its neighbors it sends a parent request to its remaining neighbor. In this way the tree grows from the leafs to a root. If a node has received parent requests from all its neighbors, it knows that it has been elected as the root of the tree. It is possible that at the end of tree identify phase two nodes send parent request messages to each other; this situation is called *root contention*, cf. Fig. 19. To resolve this situation, a root contention protocol is run. After completion of this protocol, one of the two nodes has become root of the network.

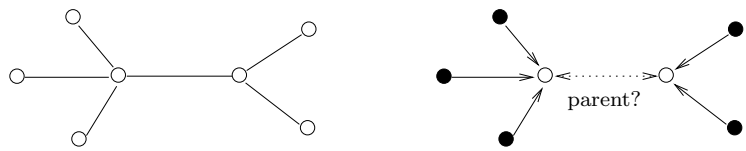


Fig. 19. Initial network topology (a) and root contention (b)

Root contention protocol. The protocol roughly works as follows. Suppose that the two contending nodes are node 0 and node 1. When node i ($i = 0, 1$) has detected root contention it first flips a coin. If head comes up it waits a short period of time, somewhere in the interval $[\delta_{fast}, \Delta_{fast}]$; otherwise, it waits a long period of time somewhere in the interval $[\delta_{slow}, \Delta_{slow}]$. It is assumed that

$$0 \leqslant \delta_{fast} \leqslant \Delta_{fast} < \delta_{slow} \leqslant \Delta_{slow}$$

If after the waiting period no message has been received from the contender, the node sends a request message to its contender and declares itself to be a child, i.e., it assumes the contender to become a leader. Otherwise, it acknowledges the receipt of the message and declares itself to be the leader. If a node that has sent a request subsequently receives a request, then it concludes that there is a root contention again, and the protocol restarts.

The basic idea behind the protocol is that if the outcomes of the coin flips are different, the node with outcome tail (i.e., the slow one) will become root. And since with probability one the outcomes of the two coin flips will eventually be different, the root contention protocol will terminate (with probability one). This has been formally proven in [96] by manually verifying a model of the protocol in timed probabilistic I/O automata. An automatic verification of the root contention protocol has recently been reported [94], while parameterized verifications have been considered in [8,60].

8.2 Compositional Specification of the Root Contention Protocol

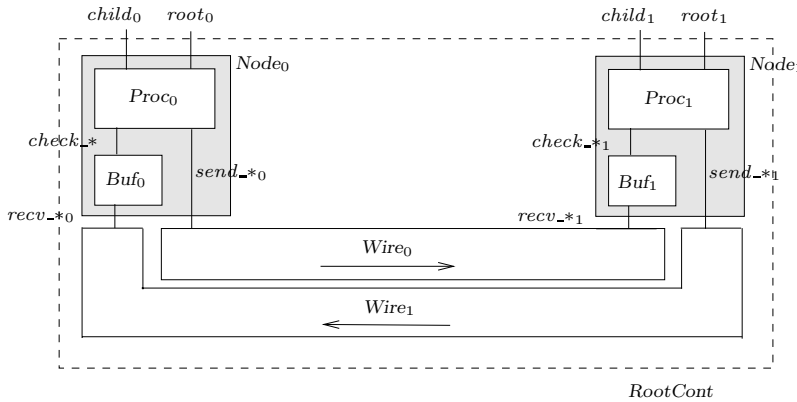


Fig. 20. Overview of the IEEE 1394 specification in

Fig. 21 presents the specification of the root contention protocol in . The model itself is based on [96]. A schematic overview of the processes involved and their synchronizations is given in Fig. 20. The $Node_i$ processes are connected to each other by two $Wire_i$ processes, that represent the communication lines between the components. Each $Node_i$ process has a Buf_i process which can hold a single message from the other $Node_{(1-i)}$. New messages from $Node_{(1-i)}$ will simply overwrite the old message. Both nodes start (via $Proc_i$) to wait $F_{x_i}()$ units of time. If after waiting, the buffer is still empty (i.e. $check\_emp_i$), the node will send a $send\_req_i$ to its contender and will subsequently wait for an

acknowledgement. If this acknowledgement (i.e. $check\_ack_i$) arrives, $Node_i$ will declare itself a child using action $child_i$. On the other hand, if after waiting $F_{x_i}()$ units of time, $Node_i$ receives a $check\_req_i$ action, it declares itself to be the leader using action $root_i$. The delay of the communication line is modelled by clock y_i.

$$RootCont = (Node_0 \,||_{\varnothing}\, Node_1) \,||_A\, (Wire_0 \,||_{\varnothing}\, Wire_1)$$

with the following process definitions ($i = 1, 2$):

$$\begin{array}{lcl}
Node_i & = & (Proc_i \,||_B\, Buf_i) \\
Proc_i & = & \mathsf{set}\ x_i\ \mathsf{in}\ (\mathsf{when}\ x_i \mapsto\ (check_emp_i; SndReq_i + check_req_i; SndAck_i)) \\
SndReq_i & = & send_req_i; (check_req_i; Proc^i + check_ack_i; child_i; \mathbf{0}\ ) \\
SndAck_i & = & send_ack_i; root_i; \mathbf{0} \\
Buf_i & = & check_emp_i; Buf_i + recv_req^i; BufReq_i + recv_ack^i; BufAck_i \\
BufReq_i & = & check_req_i; Buf_i + recv_req^i; BufReq_i + recv_ack^i; BufAck_i \\
BufAck_i & = & check_ack_i; Buf_i + recv_req^i; BufReq_i + recv_ack^i; BufAck_i \\
Wire_i & = & send_req^i; WireReq_i + send_ack_i; WireAck_i \\
WireReq_i & = & \mathsf{set}\ y_i\ \mathsf{in}\ (\mathsf{when}\ y_i \mapsto\ recv_req_{(1-i)}; Wire_i) + Wire_i \\
WireAck_i & = & \mathsf{set}\ y_i\ \mathsf{in}\ (\mathsf{when}\ y_i \mapsto\ recv_ack_{(1-i)}; Wire_i) + Wire_i
\end{array}$$

and the following synchronization sets:

$$\begin{array}{lcl}
A & = & \{\, send_req_i, send_ack_i, recv_req_i, recv_ack_i \,\} \\
B & = & \{\, check_emp_i, check_req_i, check_ack_i \,\}
\end{array}$$

Fig. 21. ⌂ specification of the root contention protocol

The clock distributions. The delay after detection of a root contention in $Node_i$ is governed by clock x_i ($i = 0, 1$) whose distribution is given by:

$$F_{x_i}(t) = \begin{cases}
0 & \text{if } t < \delta_{fast} \\
\frac{1}{2} \cdot \left(\frac{t - \delta_{fast}}{\Delta_{fast} - \delta_{fast}} \right) & \text{if } \delta_{fast} \leqslant t < \Delta_{fast} \\
\frac{1}{2} & \text{if } \Delta_{fast} \leqslant t < \delta_{slow} \\
\frac{1}{2} \cdot \left(1 + \frac{t - \delta_{slow}}{\Delta_{slow} - \delta_{slow}} \right) & \text{if } \delta_{slow} \leqslant t < \Delta_{slow} \\
1 & \text{if } \Delta_{slow} \leqslant t
\end{cases}$$

This distribution corresponds to first choosing either a short delay in the interval $[\delta_{fast}, \Delta_{fast}]$ with probability $\frac{1}{2}$, or choosing a long delay in the interval $[\delta_{slow}, \Delta_{slow}]$ with probability $\frac{1}{2}$. In fact, F_{x_i} is the combination of uniform distributions over the two respective intervals $[\delta_{fast}, \Delta_{fast}]$ and $[\delta_{slow}, \Delta_{slow}]$. The IEEE 1394 standard [62] only specifies the lower- and upper-bounds of these

intervals, but states nothing about mean, variances, or the like. Approximating these non-deterministic intervals in the above way is the most inderminate approximation conform to the principle of maximizing the entropy, see Section 2.2. A pictorial representation of F_{x_i} is given in Fig. 22.

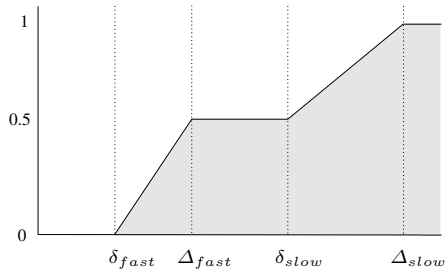


Fig. 22. Distribution of delay after detection of root contention

The transmission delay of $Wire_i$ is determined by clock y_i. This delay depends on the length of the cable. We assume that the transmission delay is between $v/2$ and a maximal velocity v with an average of $\frac{4}{5}v$. The transmission delay is approximated by a beta-distribution with parameters $\beta_1 = 2$ and $\beta_2 = 6$. To be more precise, for cable-length m, the probability density function of clock y_i $i = 0, 1)$, is given by:

$$f_{y_i}(t) \;=\; \begin{cases} \dfrac{\left(\frac{v}{m}t - 1\right)\left(2 - \frac{v}{m}t\right)^5}{\beta(2,6)} & \text{if } \frac{m}{v} \leqslant t < 2\frac{m}{v} \\ 0 & \text{if } t < \frac{m}{v} \;\text{ or }\; 2\frac{m}{v} \leqslant t \end{cases}$$

where β is a function that is well-known from statistics and probability theory, see e.g., [93].

8.3 The Time until Contention Resolution

Simulation parameters. The verification study in [96] revealed that the correctness of the root contention protocol – a leader is eventually elected with probability one – is only guaranteed if the maximum transmission delay is smaller than δ_{fast} and $(\delta_{slow} - \Delta_{fast})/2$. In our case, the maximum transmission delay is assumed to be the lowest upper-bound of the support set of F_{y_i}, i.e., $2m/v$. This corresponds to the distance the message may travel divided by the slowest speed. To guarantee the correctness of the protocol we thus assume:

$$\frac{2\,m}{v} \;<\; \min\left\{\delta_{fast}, \frac{\delta_{slow} - \Delta_{fast}}{2}\right\}$$

For $v = 198$ $mtr/\mu sec$, the maximum velocity according to the IEEE 1394 standard, we obtain $m < \min\{99 \cdot \delta_{fast}, 49.5 \cdot (\delta_{slow} - \Delta_{fast})\}$. For the discrete-event

simulation we have taken the values of δ_{fast}, Δ_{fast}, δ_{slow}, and Δ_{slow} from the specifications in the IEEE 1394 standard [62] and the draft IEEE 1394a proposal [63], cf. Table 6.

	δ_{fast}	Δ_{fast}	δ_{slow}	Δ_{slow}	m
IEEE 1394	0.24 μsec	0.26 μsec	0.57 μsec	0.60 μsec	< 15.345 mtr
IEEE 1394a	0.76 μsec	0.80 μsec	1.60 μsec	1.64 μsec	< 39.6 mtr

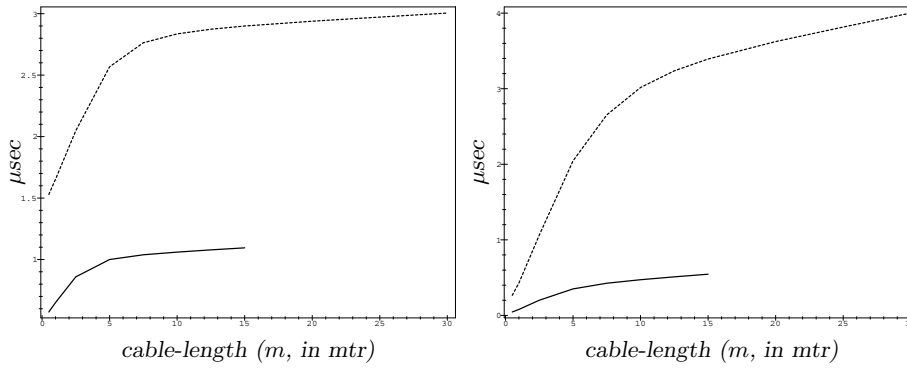
Table 6. Parameters of the root contention protocol

Simulation results. The closed behavior of the compositional specification is deterministic (with probability 1). Thus, no adversary is needed to resolve any non-determinism. The discrete-event simulation has been carried out by a prototype implementation in the functional programming language Haskell; for more details see [33]. For each setting, five series of 250,000 simulations have been carried out, each starting with different seeds for the random number generator. For IEEE 1394, the length m of the cable ranged to up to 15 meters; for IEEE 1394a, lengths of up to m=30 meters were considered. The average and variance of the time until contention resolution are reported in Fig. 23 where the lower curves refer to IEEE 1394 while the upper curves refer to IEEE 1394a. As expected, the time increases on increasing cable length m. It appears that the new (draft) version of the protocol shows a lower performance. For a given cable length, the average time until contention resolution for IEEE 1394a is about a factor two higher than for standardized IEEE 1394. This phenomenon is due to the longer delay after flipping a coin. It should be remarked, though, that the IEEE 1394a protocol guarantees the correctness of the root contention protocol over longer distances.

9 Related Work

In this section we give an overview of stochastic process algebras that incorporate general distributions and compare these proposals to ⏢.

– TIPP [43] is the earliest approach addressing general distributions in a process algebra. Its syntax has the integrated prefix $a_F; p$ which in ⏢ corresponds to set x_F in when $x_F \mapsto a; p$. Its semantics is based on labelled transition systems in which transitions are decorated with the associated distribution function and, to keep track of the execution of parallel processes, a number that indicates how many times an action has not been chosen to execute. This number introduces infinite semantic objects, even for simple regular processes.



(a) Average (b) Variance

Fig. 23. Time until contention resolution in IEEE 1394 (lower curve) and IEEE 1394a (upper curve)

- [1] extends LOTOS with several kinds of (stochastic) timers. The semantics is given in terms of a (variety of) timed transition system, therefore abstracting from probabilities. Separately, the authors consider the stochastic model as a kind of GSMP. This work proposed very interesting ingredients at language level. However, the treatment is not algebraic – despite the use of structured operational semantics. The authors impose strong syntactic restrictions, do not provide an adequate equivalence relation and, as a consequence, neither algebraic laws.
- [45,46,97] introduced a process algebra for discrete event simulation. The concerns of randomly setting a timer, expiration of such a timer, and actual activity are split in a rather similar way to ours. The semantic model is similar to our probabilistic transition systems where non-deterministic transitions are instead split into discrete transitions and timed transitions. As a consequence, semantic objects contain usually uncountably many states and transitions. Their process algebra includes an urgent and a delayable prefixing, so its interpretation combines both views of closed and open system.
- [18] studies a semantics for a process algebra similar to TIPP in terms of a stochastic extension of event structures. This model seems to be more natural to deal with general distributions since activities that are not causally dependent (i.e. concurrent activity) are not related in the model, contrarily of what occurs in interleaving based models. However, recursive processes always have associated an infinite semantic object. Recent investigations indicate, however, that finite objects can be obtained (for finite-state process algebra terms) from which stochastic task graph models are generated that can be analyzed numerically [87].
- [83] followed an approach similar to [43]. The author gives semantics to a stochastic extension of the π-calculus. In this case transitions are decorated with locality information to keep track which process performed it.

- A general semi-Markovian process algebra based on EMPA is discussed in [16]. Terms in this process algebra have semantics in a semi-interleaving model that allows for refinement of actions (the so-called ST-semantics). Although this calculus preserves finiteness of semantic objects in a reasonable way (like ours), the manner in which semantics is given is not straightforward. A nice characteristic of this process algebra is that it represents the GSMP model in a complete way.
- Recently, [17] adapted the previous work to consider —like in our case— the separation between beginning of a delay, ending of it, and execution of an action. In so doing, the authors obtained a neater semantics as well as a complete axiomatization for the weak equivalence relation they consider.
- Another recent work includes [74]. In this paper the authors pursue ideas similar to [43,83]. To keep track of the time that has passed, the transition is labelled with the action, its respective delay distribution, and the time it takes. Like [1], they also use an "age" function in order to update the distribution of the remaining delay of the concurrent events. The fact of labelling the transition with the occurrence time makes the semantic object infinite.

Among the stochastic process algebras enumerated above, [45,46,97] is the closest to  . Like , [45,46,97], [49], and [17] allow non-determinism. In all the other cases (including the Markovian process algebras), choice is always solved either by the race policy, by the pre-selection policy, or by a combination of both.

Other works that relate discrete-event simulation models (or languages) to process algebra are [82,10]. In these works, the approach is different: rather than generating a simulation model automatically from a process algebra specification (as in this paper), they use process algebra as a semantic model for simulation languages. In addition, these works do not take probabilistic timing into consideration.

10 Epilogue

In this tutorial, we have given an informal overview of incorporating general distributions in a process algebraic framework. We discussed its complications and gave an overview of possible solutions that have been suggested so far in the literature. We recapitulated the model of generalized semi-Markov chains (GSMCs) and justified the need for a process algebraic treatment of such models. In the second part of the paper, we have presented stochastic automata, a model that subsumes GSMCs, includes non-determinism and is amenable to composition. A process algebra named has been presented to describe these stochastic automata in a modular way. As a result, stochastic automata (and thus GSMCs) can be generated automatically from specifications. Analysis methods for specifications have been treated. The proposed approach has been illustrated by specifying the root contention phase within the standardized IEEE 1394 serial bus protocol. As opposed to other studies of this protocol that focus on its func-

tional correctness we have studied the delay until root contention resolution. Below we conclude by listing some interesting research topics for future work.

Weak bisimulation. Thanks to the efforts of several research groups, most semantic issues of incorporating general distributions into a process algebraic setting have been satisfactorily solved. One of the most important (semantic) issues for future work is the investigation of weak equivalence relations such as variants of weak or branching bisimulation. A weak equivalence relation that allows to abstract from immediate invisible actions has been defined recently [17]. To our knowledge, the replacement of a sequence of generally distributed delays by a single delay labelled with the convolution of the delays on the sequence (i.e., the sum of the random variables involved), has not yet been captured in a congruence relation.

Model checking. As an alternative to discrete-event simulation techniques, model checking could be applied to stochastic automata. Using such techniques one would be able to check properties of the form "with probability at most 0.99 the system will deadlock within 2 hours" in a fully automated way. Such quantitative model checking algorithms have been recently considered for models that are closely related to stochastic automata: for probabilistic variants of the duration calculus on continuous semi-Markov processes [61] and for a continuous probabilistic variant of timed automata [69]. It would be interesting to see how these techniques can be effectively applied to stochastic automata and  specifications. Besides, the combination of these techniques with discrete-event simulation could be investigated.

User-oriented specification languages. Performance engineers do consider process algebra as being (too) complicated. In order to bridge the gap towards practitioners that e.g. use stochastic automata for modeling real-time multi-media systems [11,20], efforts should be made towards making the specification formalism more user-friendly. Interesting developments in that respect are the usage of stochastic automata for stochastic extensions of UML (Universal Modeling Language) Statecharts [42] and the activities on MoDeST (a Model Description language for Stochastic Timed Systems) that builds upon  , Promela [59], and timed automata and for which tool-support is currently under development.

Acknowledgements

Ed Brinksma, Holger Hermanns, Ulrich Herzog, Ric Klaren, Rom Langerak, Diego Latella, Mieke Massink and Theo Ruys are all kindly acknowledged for their support and their joint experiences with us over the last years on the research on general distributions in process algebra. We thank Boudewijn Haverkort for providing us with information concerning entropies. The second author is supported by the Dutch Technology Foundation (STW) on the PROGRESS project HaaST (Verification of Hard and Softly Timed Systems).

References

1. M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto and A. Valenzano. A LOTOS extension for the performance analysis of distributed systems. *IEEE/ACM Trans. on Networking*, **2**(2), 151–164, 1994.
2. R. Alur and D.L. Dill. A theory of timed automata. *Th. Comp. Sc.*, **126**: 183–235, 1994.
3. S. Asmussen, O. Nermand and M. Olsson. Fitting phase-type distributions via the EM algorithm. *Scand. J. Statist.*, **23**: 420–441, 1996.
4. J. Baeten (ed). *Applications of Process Algebra*. Cambridge Tracts in Theoretical Computer Science, Cambridge Univ. Press, 1990.
5. J. Baeten and J. Bergstra. Real time process algebra. *Form. Asp. of Comp.*, **3**(2):142–188, 1991.
6. J. Baeten, J. Bergstra and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Inf. & Comp.*, **121**: 234–255, 1995.
7. J. Baeten and C. Verhoef. A congruence theorem for structured operational semantics. In E. Best, ed, *Concurrency Theory*, LNCS 715: 477–492, Springer, 1993.
8. G. Bandini, R. Lutje Spelberg, and W.J. Toetenel. Parametric verification of the IEEE 1394a root contention protocol using LPMC, 2000 (submitted).
9. M. Bernardo and R. Gorrieri. A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Th. Comp. Sc.*, **202**: 1–54, 1998.
10. G. Birtwistle and C. Tofts. A denotational semantics for a process-based simulation language. *ACM Trans. on Modeling and Computer Simulation*, **8**(3): 281–305, 1998.
11. L. Blair, T. Jones and G. Blair. Stochastically enhanced timed automata. In S.F. Smith and C.L. Talcott, eds, *Proc. IFIP Conf. on Formal Methods for Open Object-based Distributed Systems IV*, pp. 327–350, Chapman & Hall, 2000.
12. H. Bohnenkamp and B.R. Haverkort. Stochastic event structures for the decomposition of stochastic process algebra models. In J. Hillston and M. Silva, eds, *Proc. 7th Workshop on Process Algebras and Performance Modelling*. Prensas Universitarias de Zaragoza, 1999.
13. T. Bolognesi, F. Lucidi and S. Trigila. Converging towards a timed LOTOS standard. *Comp. Standards & Interfaces*, **16**: 87–118, 1994.
14. S. Bornot, J. Sifakis and S. Tripakis. Modeling urgency in timed systems. In: W.-P. de Roever, H. Langmaack and A. Pnueli, eds, *Compositionality: The Significant Difference*, LNCS 1536: 103–129. Springer, 1998.
15. A. Bouajjani, S. Tripakis and S. Yovine. On-the-fly symbolic model-checking for real-time systems. In *Proc. IEEE Real-Time Systems Symp.*, pp. 25–34, IEEE CS Press, 1997.
16. M. Bravetti, M. Bernardo and R. Gorrieri. Towards performance evaluation with general distributions in process algebras. In D. Sangiorgi and R. de Simone, eds, *Concurrency Theory*, LNCS 1466: 405–422, Springer, 1998.
17. M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-Markov processes. *Th. Comp. Sc.*, **286**, 2001 (to appear).
18. E. Brinksma, J.-P. Katoen, R. Langerak and D. Latella. A stochastic causality-based process algebra. *The Comp. J.*, **38**(7): 552–565, 1995.
19. E. Brinksma and H. Hermanns. Process algebra and Markov chains. This volume.

20. J. Bryans, L. Blair, H. Bowman, and J. Derrick. Specification and analysis of automata-based designs. In W. Grieskamp, T. Santen and B. Stoddart, eds, *Integrated Formal Methods*, LNCS 1945: 176–193, 2000.
21. C.G. Cassandras. *Discrete Event Systems – Modeling and Performance Analysis.* Irwin Inc. and Aksin Associates Inc., 1993.
22. G. Clark. Stochastic process algebra structure for insensitivity. In J. Hillston and M. Silva, eds, *Proc. 7th Workshop on Process Algebras and Performance Modelling*, pp. 63–82. Prensas Universitarias de Zaragoza, 1999.
23. G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models.* PhD thesis, University of Edinburgh, 2000.
24. R.W. Cleaveland and S.A. Smolka. Strategic directions in concurrency research. *Computing Surveys*, **28**(4): 607–625, 1996.
25. D.R. Cox. A use of complex probabilities in the theory of stochastic processes. *Proc. Cambridge Philosophy Society*, **51**: 313–319, 1955.
26. M.E. Crovella. Performance evaluation with heavy tailed distributions (extended abstract). In B. Haverkort, H. Bohnenkamp and C. Smith, eds, *Computer Performance Evaluation*, LNCS 1786: 1–9, Springer, 2000.
27. P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, University of Twente, 1999.
28. P.R. D'Argenio. A compositional translation of stochastic automata into timed automata. CTIT Technical Report 00-08, 32 pp., 2000.
29. P.R. D'Argenio and E. Brinksma. A calculus for timed automata (extended abstract). In B. Jonsson and J. Parrow, eds, *Formal Techniques in Real-Time and Fault Tolerant Systems*, LNCS 1135, pp. 110-129, Springer, 1996.
30. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. A stochastic automata model and its algebraic approach. In E. Brinksma and A. Nymeyer, eds, *Proc. 5th Int. Workshop on Process Algebra and Performance Modelling*, CTIT Technical Report 97-14, pp. 1–16, 1997.
31. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, eds, *Proc. IFIP Conf. on Programming Concepts and Methods.* Chapman & Hall, pp. 126–147, 1998.
32. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. A compositional approach to generalised semi-Markov processes. In A. Guia, M. Spathopoulos and R. Smedinga, eds, *Proc. 4th Int. Workshop on Discrete-Event Systems*, pp. 391–397. IEE Press, 1998.
33. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. Specification and analysis of soft real-time systems: Quantity and quality. In *Proc. IEEE Real-Time Systems Symp.*, pp. 104–114, IEEE CS Press, 1999.
34. C. Derman. *Finite-State Markovian Decision Processes.* Academic Press, 1970.
35. M. Devillers, D. Griffioen, J. Romijn and F. Vaandrager. Verification of a leader election protocol: formal methods applied to IEEE 1394. *Form. Methods in Sys. Design*, **16**(3): 307–320, 2000.
36. J. Engelfriet. Branching processes of Petri nets. *Acta Inf.*, **28**: 575–591, 1991.
37. D. Ferrari. Considerations on the insularity of performance evaluation. *IEEE Trans. on Soft. Eng.*, **12**(6): 678–683, 1986.
38. A. Feyzi Ates, M. Bilgic, S. Saito, and B. Sarikaya. Using timed CSP for specification, verification and simulation of multimedia synchronization. *IEEE J. on Sel. Areas in Comm.*, **14**:126–137, 1996.
39. W. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science, Springer, 2000.

40. R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, eds, *PARLE — Parallel Architectures and Languages Europe*, LNCS 259: 224–242. Springer, 1987.
41. P.W. Glynn. A GSMP formalism for discrete event simulation. *Proc. of the IEEE*, **77**(1):14–23, 1989.
42. S. Gnesi, D. Latella and M. Massink. A stochastic extension of a behavioural subset of UML statechart diagrams. In: Proc. High Assurance System Engineering, IEEE CS Press, 2000.
43. N. Götz, U. Herzog and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In L. Donatiello and R. Nelson, eds, *Performance Evaluation of Computer and Communication Systems*, LNCS 729: 121–146. Springer, 1993.
44. H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proc. 11th IEEE Real-Time Systems Symposium*, pp. 278–287, 1990.
45. P.G. Harrison and B. Strulo. Stochastic process algebra for discrete event simulation. In F. Baccelli, A. Jean-Marie and I. Mitrani, eds, *Quantitative Methods in Parallel Systems*, pp. 18–37. Springer, 1995.
46. P.G. Harrison and B. Strulo. SPADES: a stochastic process algebra for discrete event simulation. *J. of Logic and Computation*, **10**(1): 3–42, 2000.
47. C. Harvey. Performance engineering as an integral part of system design. *Br. Telecom Technol. J.*, **4**(3): 142–147, 1986.
48. W. Henderson and D. Lucic. Aggregation and disaggregetion through insensitivity in stochastic Petri nets. *Perf. Ev.*, **17**: 91–114, 1993.
49. H. Hermanns. *Interactive Markov Chains.* PhD. thesis, U. Erlangen-Nürnberg, 1998.
50. H. Hermanns, U. Herzog and J.-P. Katoen. Process algebra for performance eveluation. *Th. Comp. Sci.*, 2001 (to appear).
51. H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Sci. of Comp. Programming*, **36**(1): 97–127, 2000.
52. H. Hermanns and M. Rettelbach. Towards a superset of LOTOS for performance prediction. In [84], pp. 77–94.
53. U. Herzog. A concept for graph-based stochastic process algebras, generally distributed activity times, and hierarchical modelling. In [84], pp. 1–20.
54. D.P. Heyman and M.J. Sobel. *Stochastic Models in Operations Research, volume 1 - Stochastic Processes and Operating Characteristics.* McGraw-Hill, 1982.
55. J. Hillston. On the nature of synchronisation. In U. Herzog and M. Rettelbach, eds, *Proc. 2nd Int. Workshop on Process Algebra and Performance Modelling*, Arbeitsbericht **27**(4): 51–71, University of Erlangen, 1994.
56. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
57. J. Hillston and M. Ribaudo. Stochastic process algebras: a new approach to performance modeling. In K. Bagchi and G. Zobrist, eds, *Modeling and Simulation of Advanced Computer Systems.* Gordon Breach, 1998.
58. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
59. G. Holzmann. The model checker SPIN. *IEEE Trans. on Softw. Eng.*, **23**(5), 279–295, 1997.

60. T. Hune, J. Romijn, M.I. Stoelinga and F. Vaandrager. Linear parametric model checking of timed automata. In: T. Margaria and W. Yi, eds, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2031: 189–204. Springer, 2001.
61. D. van Hung and Z. Chaochen. Probabilistic duration calculus for continuous time. *Formal Aspects of Computing*, **11**: 21–44, 1999.
62. IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, 1996.
63. IEEE Computer Society. P1394a Draft Standard for a High Performance Serial Bus (Supplement). Draft 2.0, 1998.
64. ISO IS 8807. LOTOS – A formal description technique based on the temporal ordering of observational behaviour. 1989.
65. B. Jonsson, W. Yi and K.G. Larsen. Probabilistic extensions of process algebras. In J. Bergstra, A. Ponse and S.A. Smolka, eds, *Handbook of Process Algebra*, North-Holland, pp. 685–710, 2001.
66. K. Kanani. *A Unified Framework for Systematic Quantitative and Qualitative Analysis of Communicating Systems.* PhD thesis, Imperial College, Univ. of London, 1998.
67. J.-P. Katoen, E. Brinksma, D. Latella and R. Langerak. Stochastic simulation of event structures. In [84], pp. 21–40.
68. J. Kemeny and J. Snell. *Finite Markov Chains.* Van Nostrand, 1960.
69. M.Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying Quantitative properties of continuous probabilistic timed automata. In C. Palamadessi, ed, *Concurrency Theory*, LNCS 1877: 123–137. Springer, 2000.
70. S. Lang. *Real and Functional Analysis*, volume 142 of *Graduate Texts in Mathematics.* Springer, 1993.
71. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. and Comp.*, **94**(1): 1–28, 1992.
72. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. J. on Softw. Tools for Technol. Transf.*, **1**(1/2): 134–152, 1997.
73. L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Comp. Netw. & ISDN Sys.* **29**(3): 271–292, 1997.
74. N. Lopez and M. Núnez. NMSPA: A non-Markovian model for stochastic processes. In *Proc. 1st Int. Workshop on Distributed System Validation and Verification*, IEEE CS Press, 2000.
75. G. Lowe. Probabilistic and prioritized models of timed CSP. *Th. Comp. Sci.*, **138**: 315–352, 1995.
76. C. Miguel, A. Fernández and L. Vidaller. LOTOS extended with probabilistic behaviours. *Form. Asp. of Comp.*, **5**: 253–281, 1993.
77. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
78. F. Moller and C. Tofts. A temporal calculus of communicating systems. In J. Baeten and J-W. Klop, eds, *Concur'90: Theories of Concurrency – Unification and Extension*, LNCS 458: 401–415. Springer, 1990.
79. M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models – An Algorithmic Approach.* The Johns Hopkins University Press, 1981.
80. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In J.W. de Bakker et al, eds, *Real-Time: Theory in Practice*, LNCS 600: 526–548. Springer, 1992.
81. M. Nielsen, G.D. Plotkin and G. Winskel. Petri nets, event structures and domains. *Th. Comp. Sci.*, **13**(1):85–108, 1981.

82. R.J. Pooley. Integrating behavioural and simulation modelling. In *Quantitative Evaluation of Computing and Communication Systems*, LNCS 977: 102–116. Springer, 1995.
83. C. Priami. Stochastic π-calculus with general distributions. In [84], pp. 41–57.
84. M. Ribaudo, ed. *Proc. 4th Int. Workshop on Process Algebra and Performance Modelling*. C.L.U.T. Press, 1996.
85. J. Romijn. *Analysing Industrial Protocols with Formal Methods.* PhD thesis, Univ. of Twente, 1999.
86. S.M. Ross. *Stochastic Processes*. John Wiley & Sons, New York, 1983.
87. T. Ruys, R. Langerak, J.-P. Katoen, D. Latella and M. Massink. First passage time analysis of stochastic process algebra using partial orders. In: T. Margaria and W. Yi, eds, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2031: 220–235. Springer, 2001.
88. R. Schassberger. Insensitivity of steady-state distributions of GSMPs. *Ann. Probability*, **5**: 87–89, 1977.
89. S. Schneider. An operational semantics for timed CSP. *Inf. & Comp.*, **116**:193–213, 1995.
90. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic J. of Computing*, **2**(2):250–273, 1995.
91. C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in μCRL. *Form. Asp. of Comp.*, **10**: 509–531, 1998.
92. G.S. Shedler. *Regenerative Stochastic Simulation*. Academic Press, Inc., 1993.
93. A.N. Shiryaev. *Probability*. Graduate Texts in Mathematics, Springer-Verlag, 1989.
94. D. Simons and M. Stoelinga. Mechanical verification of the IEEE 1394a root contention protocol using Uppaal2k. Tech. Rep. CSI-R0009, University of Nijmegen, 2000.
95. W. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.
96. M.I. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. In J.-P. Katoen, ed, *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601: 53–74. Springer, 1999.
97. B. Strulo. *A Process Algebra for Discrete-Event Simulation*. PhD. thesis, Imperial College, U. London, 1993.
98. M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc.* 26^{th} *IEEE Symp. on Foundations of Comp. Sc.*, pp. 327–338. IEEE CS Press, 1985.
99. W. Yi. Real-time behaviour of asynchronous agents. In J. Baeten and J-W. Klop, eds, *Concur'90: Theories of Concurrency – Unification and Extension*, LNCS 458: 502–520. Springer, 1990.
100. W. Whitt. Continuity of generalized semi-Markov processes. *Math. Oper. Res.*, **5**: 494–501, 1980.

Author Index