Frontiers in Artificial Intelligence and Applications

# AN INDUCTIVE LOGIC PROGRAMMING APPROACH TO STATISTICAL RELATIONAL LEARNING

Kristian Kersting

IOS Press

# AN INDUCTIVE LOGIC PROGRAMMING APPROACH TO STATISTICAL RELATIONAL LEARNING

Frontiers in Artificial Intelligence and Applications

Volume 148

*Published in the subseries*

# Dissertations in Artificial Intelligence

*Under the Editorship of the ECCAI Dissertation Board*
*Proposing Board Member: Luc De Raedt*

*Recently published in this series*

Vol. 147. H. Fujita and M. Mejri (Eds.), New Trends in Software Methodologies, Tools and Techniques – Proceedings of the fifth SoMeT_06
Vol. 146. M. Polit et al. (Eds.), Artificial Intelligence Research and Development
Vol. 145. A.J. Knobbe, Multi-Relational Data Mining
Vol. 144. P.E. Dunne and T.J.M. Bench-Capon (Eds.), Computational Models of Argument – Proceedings of COMMA 2006
Vol. 143. P. Ghodous et al. (Eds.), Leading the Web in Concurrent Engineering – Next Generation Concurrent Engineering
Vol. 142. L. Penserini et al. (Eds.), STAIRS 2006 – Proceedings of the Third Starting AI Researchers' Symposium
Vol. 141. G. Brewka et al. (Eds.), ECAI 2006 – 17th European Conference on Artificial Intelligence
Vol. 140. E. Tyugu and T. Yamaguchi (Eds.), Knowledge-Based Software Engineering – Proceedings of the Seventh Joint Conference on Knowledge-Based Software Engineering
Vol. 139. A. Bundy and S. Wilson (Eds.), Rob Milne: A Tribute to a Pioneering AI Scientist, Entrepreneur and Mountaineer
Vol. 138. Y. Li et al. (Eds.), Advances in Intelligent IT – Active Media Technology 2006
Vol. 137. P. Hassanaly et al. (Eds.), Cooperative Systems Design – Seamless Integration of Artifacts and Conversations – Enhanced Concepts of Infrastructure for Communication
Vol. 136. Y. Kiyoki et al. (Eds.), Information Modelling and Knowledge Bases XVII
Vol. 135. H. Czap et al. (Eds.), Self-Organization and Autonomic Informatics (I)
Vol. 134. M.-F. Moens and P. Spyns (Eds.), Legal Knowledge and Information Systems – JURIX 2005: The Eighteenth Annual Conference

ISSN 0922-6389

# An Inductive Logic Programming Approach to Statistical Relational Learning

Kristian Kersting
*Faculty of Applied Sciences, Institute for Computer Science,*
*Albert-Ludwigs-Universität Freiburg im Breisgau, Germany*

IOS Press
Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

ISBN 1-58603-674-2
Library of Congress Control Number: 2006932504

*Publisher*
IOS Press
Nieuwe Hemweg 6B
1013 BG Amsterdam
Netherlands
fax: +31 20 687 0019
e-mail: order@iospress.nl

*Distributor in the UK and Ireland*
Gazelle Books Services Ltd.
White Cross Mills
Hightown
Lancaster LA1 4XS
United Kingdom
fax: +44 1524 63232
e-mail: sales@gazellebooks.co.uk

*Distributor in the USA and Canada*
IOS Press, Inc.
4502 Rachael Manor Drive
Fairfax, VA 22032
USA
fax: +1 703 323 3668
e-mail: iosbooks@iospress.com



PRINTED IN THE NETHERLANDS

April 2006


Dissertation zur Erlangung des Doktorgrades
der Fakultät für Angewandte Wissenschaften
der Albert-Ludwigs-Universität Freiburg im Breisgau

Kristian Kersting
kersting@informatik.uni-freiburg.de

Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 52, 79110 Freiburg i. Br., Germany

This page intentionally left blank

*To my family, for their love and support.*

"Ma pièce est faite, il suffit de l'écrire."

– J. B. Racine, French dramatic poet (1639-1699) –

This page intentionally left blank

*An Inductive Logic Programming Approach to Statistical Relational Learning*
*K. Kersting*
*IOS Press, 2006*


# Preface

It has been a great pleasure to be asked to write the preface for the book based on Kristian Kersting's thesis. There is no doubt in my mind that this is a remarkable and outstanding piece of work.

In his thesis Kristian has made an assault on one of the hardest integration problems at the heart of Artificial Intelligence research. This involves taking three disparate major areas of research and attempting a fusion among them. The three areas are: Logic Programming, Uncertainty Reasoning and Machine Learning. Every one of these is a major sub-area of research with its own associated international research conferences. Having taken on such a Herculean task, Kristian has produced a series of widely published results which are now at the core of a newly emerging area: Probabilistic Inductive Logic Programming. The new area is closely tied to, though strictly subsumes, a new field known as "Statistical Relational Learning" which has in the last few years gained major prominence in the American Artificial Intelligence research community.

Within his thesis Kristian makes several major contributions, many of which have already been published in refereed conference and journal papers. Firstly, Kristian introduces a series of definitions which circumscribe the new area formed by extending Inductive Logic Programming to the case in which clauses are annotated with probability values. This represents a new and powerful framework which supersedes a number of influential papers and research areas in Artificial Intelligence. Secondly, Kristian introduces Bayesian Logic Programs (BLPs). These represent an elegantly defined lifting of Judea Pearl's Bayesian networks to the logic programming level. Since Kristian's introduction of BLPs, a number of results indicate that BLPs generalise many previously defined representations, not the least of which are Bayesian networks, Logic Programs, Probabilistic Relation Models and Stochastic Logic Programs. Next Kristian investigates the approach of Learning from proofs. This is an interesting new learning framework which is the first to go beyond the two standard semantic frameworks of Inductive Logic Programming.

Kristian then looks at the problem of upgrading HMMs to logical HMMs. Hidden Markov Models (HMMs) are one of the most widely used machine learning technologies in Statistical Linguistics and Bioinformatics, and allow the representation of probabilistic finite automata. Kristian has upgraded standard HMMs to allow relational descriptions to be included within the description of the automata. The three standard HMM estimation algorithms are also upgraded. He has demonstrated the power of such representations using biological predictive modelling problems, and shown performance increases over alternative approaches.

Kristian next considers the issue of upgrading Fisher Kernels to Relational Fisher kernels. Fisher kernels have been widely used within statistics and more recently

in support vector machines. Building on his previous approaches involving lifting propositional representations Kristian shows how relations can be usefully included within this context. The approach was empirically tested on protein fold prediction and shown to have high predictive accuracy relative to logical HMMs.

Lastly, Kristian introduces Markov decision programs. As a final demonstration of his general approach Kristian shows how temporal descriptions involving action can be introduced by lifting Markov decision processes to logical Markov decision programs. Kristian demonstrates how these can be learned using relational reinforcement algorithms which he tests empirically in a Blocks World setting.

In summary, this thesis represents an extremely powerful and wide-ranging study which has made strong contributions right across the intellectual landscape. Both Kristian and his thesis supervisor Luc De Raedt, should be highly commended for this important contribution.

London, July 2006 *Stephen H. Muggleton*

# Acknowledgments

WORKING on the Ph.D. has been a wonderful and often overwhelming experience. It is hard to say whether is has been grappling with the topic itself which has been the real learning experience, or grappling with how to write papers and proposals, give talks, work in a group, stay up until the birds start singing, and stay focus ...

In any case, I am indebted to many people for making the time working on my Ph.D. an unforgettable experience.

First of all, I am deeply grateful to my advisor Luc De Raedt. To work with you has been a real pleasure to me, with heaps of fun and excitement. You have been a steady influence throughout my Ph.D. career; you have oriented and supported me with promptness and care, and have always been patient and encouraging in times of new ideas and difficulties; you have listened to my ideas and discussions with you frequently led to key insights. Your ability to select and to approach compelling research problems, your high scientific standards, and your hard work set an example. I admire your ability to balance research interests and personal pursuits. Above all, you made me feel a friend, which I appreciate from my heart.

Furthermore, I am very grateful to my external reviewer Stephen H. Muggleton, for insightful comments both in my work and in this thesis, for his support, and for many motivating discussions.

In addition, I have been very privileged to get to know and to collaborate with many other great people who became friends over the last several years. I learned a lot from you about life, research, how to tackle new problems and how to develop techniques to solve them. Niels Landwehr has been a pleasure to work with. Your technical excellence and tremendous grasp of experimental issues had a great impact on me. Over the last few years, Tapani Raiko has been a faithful friend and co-author. Thank you for teaching me so much in our joint research on logical hidden Markov models. I have greatly enjoyed the opportunity to work with Thomas Gärtner, whose mathematical insight is impressive. Many of the results on decision-theoretic reasoning would not have been possible without Alan Fern, Bob Givan, and Martijn Van Otterlo. You guys taught me a great deal about the technical groundwork. Bob's example of an independent researcher while I was visiting him at Purdue University was a valuable source of motivation. Wolfram Burgard and Rudolph Triebel gave me an understanding of probabilistic robotics. Long discussions with Kurt Driessens and Jan Ramon have significantly improved my work and inspired many new research directions. I also had the great pleasure of meeting Lise Getoor. From the very beginning of my Ph.D. career, you supported me. Thank you for the fun and the encouraging discussions while visiting you at the University of Maryland at College Park.

The work on this thesis was supported by the European Union IST programme

under contract numbers IST-2001-33053 and FP6-508861, *Application of Probabilistic Inductive Logic Programming* (APrIL) I and II. My dearest thanks to the people in this unique consortium: Björn Bringmann, Jianzhong Chen, James Cussens, Luc De Raedt, François Fages, Niels Landwehr, Heikki Mannila, Stephen H. Muggleton, Paolo Frasconi, Taneli Mielikäinen, Andrea Passerini, Hiroaki Watanabe, Sylvain Soliman, Mike Sternberg, and Sunna Torge.

I also want to thank James Cussens, whose mathematical understanding is tremendous and whose scientific work inspired me a lot. My work has greatly benefited from suggestions and kind encouragement from Manfred Jäger. Your technical depth and attention to detail, which I experienced not only while visiting you at the Max-Planck Institut für Informatik, is amazing. I am particularly thankful to David Page, who gave me, when I was a young Ph.D. student, great insights into inductive logic programming while visiting him at the University of Wisconsin–Madison. I was very lucky to have crossed paths with Hendrik Blockeel, Adnan Darwiche, Pedro Domingos, Sašo Džeroski, David Jensen, Peter Flach, Paolo Frasconi, Johannes Fürnkranz, Stephen Muggleton, Tobias Scheffer, and Ashwin Srinivasan. Thank you for your support and encouragement. I would also like to thank Joost N. Kok (editor of the FAIA series at IOS Press) for his encouragement to publish the thesis in FAIA's subseries 'Dissertations in Artificial Intelligence' (under the editorship of the ECCAI Dissertation Board), and Maarten Fröhlich and Lies Kromhout (both IOS Press) for their support during the final stages of this process.

I am also indebted to the students I had the pleasure to work with. You have been an invaluable support day in, day out, during all these years. Special thanks to Alexandru Cocora, Jörg Fischer, Steven Ganzert, Bernd Gutmann, Tayfun Guerel, Johannes Horstmann, Angelika Kimmig, Niels Landwehr, Gerrit Merkel, Livia Predoiu, Hans-Martin Schultze, and Christian Stolle. Very special thanks go to Uwe Dick, a hero that implemented Balios. Without the magic powers of the Java wizard Ingo Thon, Xanthos would not look the way it does. Thanks to all of you! This thesis would not have happened without your help.

One of the most interesting and instructive interaction I had was with the members of Luc's research group. I want to thank all 'Freiburg gang' members (present and past, Marie Curie fellows, researchers) for their support and for providing a pleasant and productive working atmosphere: Björn Bringmann, Jörg Fischer, Steven Ganzert, Gemma Casas Garriga, Tayfun Gürel, Elias Gyftodimos, Christoph Helma, Andreas Karwath, Stefan Kramer, Niels Landwehr, Sau Dan Lee, Pawel Majewski, Siegfried Nijssen, Lubomir Popelinsky, Tapani Raiko, Ulrich Rückert, Christian Stolle, Sunna Torge, Martijn Van Otterlo, and Albrecht Zimmermann.

Continuing a fine tradition, I want to thank the 'Firenze' companions Achim Brucker, Dirk Bockhorn, Felix Christopher Klaedtke, Marc Herbstritt, Noe Spinner, and Tobias Schubert for their support and the enjoyable discussions about life, Ph.D.s, politics, computer science, and all that. Furthermore, I also want to thank all other friends who put up with me through the whole Ph.D. process and helped me with personal challenges, in particular Patrick Pfaff, Rudolph and Tonio Triebel, and Alexander Tritschler. I am very grateful to Nadine Buck for her friendship over 9 years. Thank you for sharing these years with me.

I would like to thank Xenia Schnabel for her love and encouragement. And, thank

you for your support when I have needed it the most. You really let me feel a 'kleiner König'. Thank you with all my heart!

Finally, I would like to thank my mom, dad, and sister for their infinite support throughout everything. Danke für Alles. Es ist schön, daß es euch gibt.

Freiburg i. Br., April 2006 *Kristian Kersting*

This page intentionally left blank

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Example Domains

*An Inductive Logic Programming Approach to Statistical Relational Learning*
*K. Kersting*
*IOS Press, 2006*


# Abstract

Statistical relational learning addresses one of the central questions of artificial intelligence: the integration of probabilistic reasoning with first order logic representation and machine learning. Recently, this questions has received a lot of attention. Several statistical relational learning approaches have been developed in related, but different areas including machine learning, statistics, databases, and reasoning under uncertainty.

This thesis starts from an inductive logic programming perspective and firstly develops a general framework for statistical relational learning: probabilistic inductive logic programming. Based on this foundation, the thesis shows how to incorporate the logical concepts of objects and relations among these objects into Bayesian networks. As time and actions are not just other relations, it afterwards develops approaches to probabilistic inductive logic programming over time and for making complex decision in relational domains. More specifically, Bayesian networks are upgraded to Bayesian logic programs, hidden Markov models to logical hidden Markov models; and Markov decision processes to Markov decision programs. Furthermore, it will be shown that statistical relational learning approaches naturally yield kernels for structured data. The resulting approaches will be illustrated using examples from genetics, bio-informatics, and classical planning domains.

# Overture

After motivating and overviewing the thesis, the overture introduces the framework of probabilistic inductive logic programming: an inductive logic programming view on statistical relational learning.

# Introduction

*... in which statistical relational learning is illustrated, the thesis' inductive logic programming perspective on statistical relational learning is motivated, and the thesis' outline and main contributions are presented ...*

## 1.1 Statistical Relational Learning

One of the central open questions of artificial intelligence is concerned with combining expressive knowledge representation formalisms such as relational and first-order logic with principled probabilistic and statistical approaches to inference and learning, cf. Figure 1.1. The fields of knowledge representation and inductive logic programming stress the importance of relational and logical representations that provide the flexibility and modularity to model large domains. They also highlight the importance of making general statements, rather than making statements for every single aspect of the world separately. The fields of statistical learning and uncertainty in artificial intelligence emphasize that agents that operate in the real world must deal with uncertainty. An agent typically receives only noisy or limited information about the world; actions are often non-deterministic; and an agent has to take care of unpredictable events. Probability theory provides a sound mathematical foundation for inference and learning under uncertainty. Machine learning, in general, argues that an agent needs to be capable of improving its performance through experience. Thus, the combination of expressive knowledge representation with probabilistic approaches to inference and learning is needed in order to face the challenges of real-world applications, which are complex and heterogeneous. Consider for example intelligent transportation systems. The U.S. Department of Transportation estimates that congestion costs the country about $100 billion a year in lost productivity [Euler and Robertson, 1995]. For the 1991-1995 period, the U.S. Congress has therefore provided $827.6 million for the Intelligent Transportation Systems (ITS) program. The aim of the ITS program is to improve (reducing congestion, pollution etc.) travel on mass transit and highways by using advanced computer, communications, and sensor. One approach is to seek to reduce the number of single-occupancy-vehicles and to investigate instant ride sharing.

**Example 1.1** (Online Ride Sharing Service [1], adapted from [Resnick, 2003]) Xenia is new to instant ride-sharing. She is twenty-four and is trying to save money. Fur-

[1] For instance, the Seattle Smart Traveler project seeks to reduce the number of single-occupancy-vehicles and investigates instant ride sharing. In 2006, a ride sharing pilot project based on cell phones will be launched at the Frankfurt airport. The project is part of the 'Partner für Innovation' initiative of the German Federal Ministry of Education and Research (BMBF), several universities such as the Technical University Braunschweig, and companies such as Deutsche Lufthansa and Deutsche Post. For some existing ride sharing services, see for example `www.ridenow.org`, `www.erideshare.com`, and `www.rideshareonline.com`.

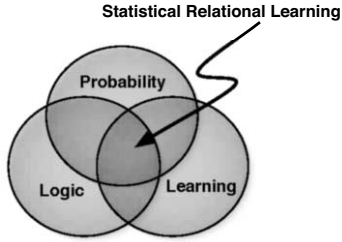



**Figure 1.1.** Statistical relational learning combines probability, logic, and learning.

thermore, it is such a hassle to park at the hospital where she works as a medical doctor. She often starts working very early in the morning and stays late at work, so she never joined a carpool. Instead, she decided to try an instant carpool system. She was a little worried about taking rides with strange men, so she set her profile to only accept rides from women, or from men who have a long history of previous rides without any complaints. She also has talked to some of her friends who also used the system. She logs onto the website and enters her profile, her address and her destination address. She also enters the list of friends who also use the systems.

The first morning, she is a little bit nervous. She walks out the door and calls the number she had pre-programmed into her cell phone. The systems tracks her progress as she walks down the main street and tells her that a green Opel Corsa is just three blocks away. After she accepts the profile of the driver — one of her friends also drove with the driver several times — her cell phone calls the driver and the car pulls up. The driver is a forty-something woman, smartly dressed with a white lab coat on the passenger seat. Sure enough, Xenia jumps in the back of the car. The driver asks Xenia what she does at the hospital and soon they discover that the driver and Xenia's boss are good friends, and the driver tells a humorous story about her boss. As they pull into a choice parking space at the hospital parking lot, reserved for multiple-occupant-vehicles, the driver smiles and says, 'You saved me 5 minutes driving around and around in this lot. Thanks! Maybe I will drive you again some time, but my schedule is very irregular so I am not sure when.'

'Thank you!' says Xenia as they walk off in different directions. As she walks away, she calls the ride sharing system again from her cell phone and presses a button to indicate that she arrived safely, that she would be happy to ride with that driver again, and that she recommends her to other passengers. ○

The ride sharing scenario is characterized by the presence of uncertainty, missing information and complex relations. There are persons, drivers, clients, parking lots, places, etc. In other words, there are multiple entities and types of entities. There is temporal information such as time of day, spatial information extracted from geographic databases such as streets, and personal constraints such as profiles, addresses or locations of people. The system guides people's choices of which driver to choose, i.e., the system requires machine learning to estimate users' preferences. That guidance is personalized, i.e., it is not only based on information (profile, feedback) from

a single person but also gathered from other people related in some way to the person (such as friends). People share rides or they walk down streets, which in turn connect places. In other words, there are multiple relations among the entities, which may change over time. The system should take advantage of information about a user's current location. It should infer when a driver brakes the optimal drive to the destination address in a way that may indicate that he has made an error, such as failing to turn left or right at a crossing. The system has to robustly track people even in the presence of total loss of cell phone signals and other sources of noise. It may predict the destination of users or provide the user with possible rides next to her current location when she is likely to need a ride. In other words, there is uncertain and missing information.

Most traditional artificial intelligence and machine learning systems, however, are able to handle either uncertainty or rich relational structures but not both. Statistical learning, reinforcement learning, and data mining methods have traditionally been developed for data in attribute-value form only. As Heikki Mannila points out in his foreword to [Džeroski and Lavrač, 2001], data is represented in matrix form: columns represent attributes, and rows represent examples. Indeed, matrices are simple and efficient matrix operations can be used. In turn, a matrix form makes it possible to devise efficient algorithms. Many — if not most — real-world data sets, however, are not in matrix form. Applications contain several entities and relationships among them. Inductive logic programming and relational learning have been developed for coping with this type of data. They do not, however, handle uncertainty in a principled way.

It is therefore not surprising that there has been a significant interest in integrating statistical learning with first order logic and relational representations. Eisele [1994] has introduced a probabilistic variant of Comprehensive Unification Formalism (CUF). In a similar manner, Muggleton [1996] and Cussens [1999] have upgraded stochastic grammars towards *stochastic logic programs*. Sato [1995] has introduced *probabilistic distributional semantics* for logic programs. Taskar et al. [2002] have upgraded Markov networks towards *relational Markov networks*, and Domingos and Richardson [2004] towards *Markov logic networks*. Another research stream includes Poole's [1993] *independent choice Logic*, Ngo and Haddawy's [1997] *probabilistic-logic programs*, Jäger's [1997] *relational Bayesian networks*, and Pfeffer's [2000] and Getoor's [2001] *probabilistic relational models*, and has investigated logical and relational extensions of Bayesian networks. This newly emerging research field is known under the name of *statistical relational learning* and *probabilistic logic learning*, see [De Raedt and Kersting, 2003] for an overview, and may be briefly defined as follows:

**Definition 1.2** (Statistical Relational Learning) Statistical relational learning deals with *machine learning* and *data mining* in *relational domains* where observations may be *missing*, *partially observed*, and/or *noisy*. ◦

Instead of giving a probabilistic characterization of logic programming such as [Ng and Subrahmanian, 1992], this line of research stresses the machine learning aspect.

Employing relational and logical abstraction within statistical learning has two advantages. First, variables, i.e., placeholders for entities allow one to make abstraction of specific entities. Second, unification allows one to share information among

entities. Thus, instead of learning regularities for each single entity independently, statistical relational learning aims at finding general regularities among groups of entities. The learned knowledge is declarative and compact, which makes it much easier for people to understand and to validate. Although, the learned knowledge must be recombined at run time using some reasoning mechanism such as backward chaining or resolution, which bears additional computational costs, statistical relational models are more flexible, context-aware, and offer — in principle — the full power of logical reasoning. Moreover, in many applications, there is a rich background theory available, which can efficiently and elegantly be represented as sets of general regularities. This is important because background knowledge often improves the quality of learning as it focuses learning on relevant patterns, i.e., restricts the search space. While learning, relational and logical abstraction allow one to reuse experience: learning about one entity improves the prediction for other entities; it might even generalize to objects, which have never been observed before. Thus, relational and logical abstraction can make statistical learning more robust and efficient. This has been proven beneficial in many fascinating real-world applications in citation analysis, web mining, web navigation, web search, natural language processing, robotics, computer vision, social network analysis, bio- and chemo-informatics, electronic games, and activity recognition. For instance, Liao et al. [2005] applied Taskar et al.'s relational Markov networks to a problem related to the ride sharing service, namely learning and inferring transportation routines.

## 1.2 Our Approach: The ILP Perspective

Whereas most of the existing works on statistical relational learning have started from a statistical and probabilistic learning perspective and extended probabilistic formalisms with relational aspects, we will take a different perspective, in which we will start from inductive logic programming (ILP) and will study how inductive logic programming formalisms, settings and techniques can be extended to deal with probabilities. This also explains the title of the thesis, '*An Inductive Logic Programming Approach to Statistical Relational Learning*'.

ILP [2] is a research field at the intersection of machine learning and logic programming [Muggleton and De Raedt, 1994]. It aims at a formal framework as well as practical algorithms for inductively learning relational descriptions (in the form of logic programs) from examples and background knowledge. However, it does not explicitly deal with uncertainty such as missing or noisy information. Dealing explicitly with uncertainty makes probabilistic ILP more powerful than ILP and, in turn, than traditional attribute-value approaches. Moreover, there are several benefits of an ILP approach to statistical relational learning. First of all, classical ILP learning settings — as we will argue — naturally carry over to the probabilistic case. The probabilistic ILP settings make abstraction of specific probabilistic relational and first order logical representations and inference and learning algorithms yielding — for the first time — general statistical relational learning settings. Second, many ILP concepts and techniques such as *more–general–than*, *refinement operators*, *least general generalization*,

[2] ILP is often also called *multi-relational data mining* (MRDM) [Džeroski and Lavrač, 2001].

and *greatest lower bound* can be reused. Therefore, many ILP learning algorithms such as Quinlan's FOIL and De Raedt and Dehaspe's CLAUDIEN can easily be adapted. Third, the ILP perspective highlights the importance of background knowledge within statistical relational learning. The research on ILP and on artificial intelligence in general has shown that background knowledge is the key to success in many applications. Finally, an ILP approach should make statistical relational learning more intuitive to those coming from an ILP background and should cross-fertilize ideas developed in ILP and statistical learning.

## 1.3 Contributions and Outline of the Thesis

The foundations of the thesis are laid in *Chapter 2* of the *Overture*, in which we formalize our ILP view on statistical relational learning called **probabilistic inductive logic programming**. After briefly reviewing logic programming concepts, we will sketch inductive logic programming and show how it can be extended to deal with probabilities. The contributions are **a novel ILP learning** setting called **learning from proofs**, the notation of a **probabilistic covers relation**, and, based on it, three different learning settings, namely, **probabilistic learning from entailment**, **from interpretations**, and **from proofs**.

Building on this foundations, the remainder of the thesis falls naturally into three parts with one intermezzo, each introducing a particular probabilistic ILP framework.

*Part I* introduces **Bayesian logic programs**. After briefly reviewing Bayesian networks, the representation language of Bayesian logic programs and their semantics are introduced in *Chapter 3*. Bayesian logic programs tightly integrate definite logic programs with Bayesian networks and, hence, define probability distributions over first-order interpretations. The key idea underlying Bayesian logic programs is to establish a one-to-one mapping between ground atoms and random variables, and between the immediate consequence operator and the dependency relation. In doing so, Bayesian logic programs combine the advantages of both definite clause logic and Bayesian networks: notions of objects and relations, a separation of quantitative and qualitative aspects of the world, and a graphical representation. We contribute several extensions of the basic Bayesian logic programming framework, which was originally introduced in [Kersting, 2000], namely a **graphical representation** for Bayesian logic programs, **methods to handle purely logical predicates** and **aggregate functions**. Then, in *Chapter 4*, we present the first **algorithm for automatically inducing Bayesian logic programs from interpretations** called **Scooby**. This includes a **definition for the likelihood** of a Bayesian logic program, **methods for estimating the parameters** of a Bayesian logic program, and an **algorithm for searching the space of candidate Bayesian logic programs** called **Scooby**. Bayesian logic programs and parts of SCOOBY have been implemented within the **Balios** system, which is briefly described after *Chapter 4*.

Many real world applications such as sequence analysis in bioinformatics require to model probability distributions over sets of sequences and trees. Bayesian logic programs, however, provide a general probabilistic ILP framework and are not customized for modeling the evolution of the environment over time. Indeed, discrete time

can be considered as yet another relation. This view, however, does not heal the curse of dimensionality: the set of possible state trajectories grows exponentially over time. In **Part II**, we introduce a novel probabilistic ILP over time approach called **logical hidden Markov model**. Logical hidden Markov models extend hidden Markov models to deal with sequences of structured symbols in the form of logical atoms. They employ logical atoms as structured (output and state) symbols. Variables in the atoms allow one to make abstraction of specific symbols. Unification allows one to share information among states. The contributions are the **representation language** and a **definition of the distribution** defined by a logical hidden Markov model in *Chapter 5*. Solutions to the three basic inference tasks: **evaluation**, **most likely state sequence**, **maximum parameter estimation** including a **definition of the likelihood** of a logical hidden Markov model are contributed in *Chapter 6*. Part II concludes by presenting the first **structural expectation-maximization algorithm for automatically inducing logical hidden Markov model from sequences of observations** called **sagEM** in *Chapter 7*. This includes **a definition of the expected likelihood** of a logical hidden Markov model given another model and an **algorithm for searching the space of candidate logical hidden Markov models**.

For Bayesian logic programs and logical hidden Markov models, we develop methods for estimating the joint distribution $\mathbf{P}(\mathbf{Z})$ of some random variables $\mathbf{Z}$. Many real world applications, however, are classification problems: One tries to estimate the dependence $\mathbf{P}(Y|\mathbf{X})$ of a target variable $Y \in \mathbf{Z}$ on some observation $\mathbf{X} = \mathbf{Z} \setminus \{Y\}$. Although classification problems can be solved with generative models it is well-known that the predictive performance of generative models estimated from a finite set of examples is often lower than that of discriminative classifiers. The *Intermezzo* in *Chapter 8* describes one of the first **discriminative (probabilistic) inductive logic programming** approaches: **relational Fisher kernels**. Relational Fisher kernels combine generative probabilistic ILP frameworks with discriminative learners. The key idea is to employ the gradient of the log likelihood of a probabilistic ILP model such as a Bayesian logic program or a logical hidden Markov model with respect to its parameters as features of a discriminative learner. The gradient captures the generative process rather than just the posterior probabilities. Thus, relational Fisher kernels employ relational and logical abstraction within discriminative learning. The contributions are **methods to compute the gradients of Bayesian logic program and logical hidden Markov models**. Relational Fisher kernels are among the first links established between statistical relational learning and kernel methods.

Finally, in *Part III*, we extend the capabilities of probabilistic ILP agents towards decision-theoretic planning. We provide the first **convergence results for relational reinforcement learning**. To do so, we introduce **Markov decision programs**, their representation language and a definition of the Markov decision process they induce in *Chapter 9*. Markov decision programs combine Markov decision processes with logic programming to reason efficiently with relational and logical axioms describing uncertain actions. Then, in *Chapter 10*, we introduce **abstract policies** and — based on them — a general scheme **generalized relational policy iteration**. We define **abstract policies** and present a general scheme for learning abstract policies, called **generalized relational policy iteration** (GRPI). Then, in the fol-

lowing, two GRPI approaches are presented: a **relational extension of temporal difference learning**, called **RTD($\lambda$)**, for the evaluation of a fixed abstract policies and a **relational value iteration** approach based on the first fully automated **relational value update operator** called **ReBel**. For both, **convergence results** are presented.

The *Finale* summarizes and concludes the thesis in *Chapters 11* and *12*. Possible future lines of research are discussed in *Chapter 13*. In the *Appendix*, we describe the logical hidden Markov models used in some experiments.

## 1.4 Citations to Previously Published Work

Some of this material has been published previously in workshop and conference papers, journal articles, and book chapters. The material in Chapter 2 appeared in [De Raedt and Kersting, 2004], which in turn was inspired on [De Raedt and Kersting, 2003]. It also includes elements of [Landwehr et al., 2005, De Raedt et al., 2005]. The material in Chapter 3 is based largely on [Kersting and De Raedt, 2005], which in turn developed from [Kersting, 2000, Kersting et al., 2000, Kersting and De Raedt, 2001b]. Chapter 4 initially appeared in [Kersting and De Raedt, 2001a,c], which were integrated in [Kersting and De Raedt, 2002]. It also includes material from [Fischer and Kersting, 2003]. The description of the Balios system builds on [Kersting and Dick, 2004]. The material in Chapters 5 and 6 is based on [Kersting et al., 2006], which in turn evolved from [Kersting et al., 2002, Raiko et al., 2002, Kersting et al., 2003b]. The material in Chapter 7 initially appeared in [Kersting et al., 2003a]. It later appeared in [Kersting and Raiko, 2005]. Some of the material in Chapter 8 can also be found in [Kersting and Gärtner, 2002, 2004, Dick and Kersting, 2006]. The material in Chapters 9 and 10 is based largely on [Kersting and De Raedt, 2003, 2004, Kersting et al., 2004]. The material in Appendix A is taken from [Kersting et al., 2006].

In the remainder of the thesis, the corresponding publications of a chapter (respectively section) are mentioned in a footnote marked by '*' at the beginning of the chapter (respectively section).

§ 2

# Probabilistic Inductive Logic Programming *

*... in which logical concepts and notations are defined and it is demonstrated how inductive logic programming (ILP) can be extended with probabilistic methods ...*

Probabilistic inductive logic programming aims at a formal framework for statistical relational learning. It extends inductive logic programming (ILP) [Muggleton and De Raedt, 1994] to explicitely deal with uncertainty. Before introducing probabilistic

* Builds mainly on [De Raedt and Kersting, 2004].

```
parent(jef,paul).                                    nat(0).
parent(paul,ann).                                    nat(s(X)) :- nat(X).
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

**Figure 2.1.** Two logic programs, *grandparent* and *nat*.

inductive logic programming, we briefly review the basic of logic programming and inductive logic programming.

## 2.1 Logic Programming Concepts

To introduce logic programs, consider Figure 2.1, containing two programs, *grandparent* and *nat*. Formally speaking, we have that `grandparent/2`, `parent/2`, and `nat/1` are **predicates** (with their *arity*, i.e., number of arguments listed explicitly). Furthermore, `jef`, `paul` and `ann` are **constants** and `X`, `Y` and `Z` are **variables**. All constants and variables are also **terms** . In addition, there exist structured terms such as `s(X)`, which contains the **functor** `s/1` of arity 1 and the term `X`. Constants are often considered as functors of arity 0. A first order **alphabet** $\Sigma$ is a set of predicate symbols, constant symbols and functor symbols. *Atoms* are predicate symbols followed by the necessary number of terms, e.g., `parent(jef,paul)`, `nat(s(X))`, `parent(X,Z)`, etc. **Literals** are atoms `nat(s(X))` (positive literal) and their negations `not nat(s(X))` (negative literals). We are now able to define the key concept of a **definite clause**. Definite clauses are formulas of the form

$$\mathtt{A} :- \mathtt{B_1}, \ldots, \mathtt{B_m}$$

where `A` and the $\mathtt{B_i}$ are logical atoms and all variables are understood to be universally quantified. For instance, the clause $c$

$$c \equiv \mathtt{grandparent(X,Y)} :- \mathtt{parent(X,Z)}, \mathtt{parent(Z,Y)}$$

can be read as `X` is the `grandparent` of `Y` if `X` is a `parent` of `Z` and `Z` is a `parent` of `Y`. We call `grandparent(X,Y)` the head($c$) of this clause, and `parent(X,Z)`, `parent(Z,Y)` the body($c$). In the remainder of the thesis, we will refer to definite clauses as **clauses**. Clauses with an empty body such as `parent(jef,paul)` are **facts**. A (definite) clause program (or **logic program** for short) consists of a set of clauses. In Figure 2.1, there are thus two logic programs, one defining `grandparent/2` and one defining `nat/1`. The set of variables in a term, atom, conjunction or clause $E$, is denoted as Var($E$), e.g., Var($c$) = {`X`, `Y`, `Z`}. A term, atom or clause $E$ is **ground** when there is no variable occurring in $E$, i.e. Var($E$) = $\emptyset$. A clause $c$ is **range-restricted** when all variables in the head of the clause also appear in the body of the clause, i.e., Var(head($c$)) $\subseteq |Vars(\text{body}(c))$.

A **substitution** $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$, e.g. {`Y`/`ann`}, is an assignment of terms $t_i$ to variables $V_i$. Applying a substitution $\theta$ to a term, atom or clause $e$ yields the instantiated term, atom, or clause $e\theta$ where all occurrences of the variables $V_i$ are simultaneously replaced by the term $t_i$, e.g. $c\theta$ is

$$c' \equiv \texttt{grandparent(X, ann)} :- \texttt{parent(X, Z), parent(Z, ann)} .$$

A substitution $\theta$ is the **most general unifier** $\text{mgu}(a, b)$ of atoms $a$ and $b$ if and only if $a = b\theta$ and for each substitution $\theta'$ such that $a = b\theta'$, there exists a substitution $\gamma$ such that $\theta' = \theta\gamma$.

A clause $c_1$ $\theta$-subsumes [3] a clause $c_2$, denoted as $c_2 \preceq_\theta c_1$, if and only if $\{\text{head}(c_2)\theta\} \cup \text{body}(c_2)\theta \subset \{\text{head}(c_1)\} \cup \text{body}(c_1)$. For instance, $\texttt{p(X)} :- \texttt{q(X)}$ subsumes $\texttt{p(X)} :- \texttt{q(X)}, \texttt{r(Y)}$. $\theta$-subsumption is reflexive and transitive, but not antisymmetric as $\texttt{p(X)} :- \texttt{q(X)}$ and $\texttt{p(X)} :- \texttt{q(X)}, \texttt{q(Y)}$ show. Thus, $\theta$-subsumption defines a pre-order on the set of clauses, i.e., a partially ordered set of equivalence classes. We say that a clause is **reduced** if it does not $\theta$-subsume any of its subclauses. Every equivalence class contains a reduced clause that is unique up to variable renaming. The set of equivalence classes forms a lattice, i.e., two clauses have a unique least upper bound and a greater lower bound under $\theta$-subsumption. The **least general generalization** (least upper bound) of two conjunctions (clauses) under ($\theta$-)subsumption is called lgg and is the least general conjunction (clause) that is subsumed by both conjunctions (clauses). The **greatest lower bound** (glb) of two conjunctions (clauses) $A$ and $B$ is the most general conjunction (clause) that is subsumed by both $A$ and $B$.

The **Herbrand base** of a logic program $P$, denoted as $\text{hb}(P)$, is the set of all ground atoms constructed with the predicate, constant and function symbols in the alphabet of $P$.

**Example 2.1** The Herbrand bases of the *nat* and *grandparent* logic programs are $\text{hb}(nat) = \{\texttt{nat(0)}, \texttt{nat(s(0))}, \texttt{nat(s(s(0)))}, ...\}$ and

$$\begin{aligned}\text{hb}(grandparent) = \{&\texttt{parent(ann, ann)}, \texttt{parent(jef, jef)},\\ &\texttt{parent(paul, paul)}, \texttt{parent(ann, jef)}, \texttt{parent(jef, ann)}, ...,\\ &\texttt{grandparent(ann, ann)}, \texttt{grandparent(jef, jef)}, ...\}.\end{aligned}$$

○

A **Herbrand interpretation** for a logic program $P$ is a subset of $\text{hb}(P)$. A Herbrand interpretation $I$ is a **model** of a clause $c$ if and only if for all substitutions $\theta$ such that $\text{body}(c)\theta \subseteq I$ holds, it also holds that $\text{head}(c)\theta \in I$. The interpretation $I$ is a model of a logic program $P$ if $I$ is a model of all clauses in $P$. A clause $c$ (logic program $P$) **entails** another clause $c'$ (logic program $P'$), denoted as $c \models c'$ ($P \models P'$), if and only if, each model of $c$ ($P$) is also a model of $c'$ ($P'$). Clearly, if clause $c$ (program $P$) $\theta$-subsumes clause $c'$ (program $P'$) then $c$ ($P$) entails $c'$ ($P'$), but the reverse is not true.

The **least Herbrand model** $\text{LH}(P)$, which constitutes the semantics of the logic program $P$, consists of all facts $f \in \text{hb}(P)$ such that $P$ logically entails $f$, i.e. $P \models f$. Various methods exist to compute the least Herbrand model. We merely sketch its computation through the use of the **immediate consequence** operator $T_P$. The

[3] The definition of $\theta$-subsumption also applies to conjunctions of literals, as these can also be defined as set of literals.

operator $T_P$ is the function on the set of all Herbrand interpretations of $P$ such that for any such interpretation $I$ we have

$$T_P(I) = \{\mathtt{A}\theta \mid \text{there is a substitution } \theta \text{ and a clause } \mathtt{A} : -\mathtt{A_1}, \ldots, \mathtt{A_n} \text{ in } P \text{ such that } \mathtt{A}\theta\text{: } -\mathtt{A_1}\theta, \ldots, \mathtt{A_n}\theta \text{ is ground and for } i = 1, \ldots, n : A_i\theta \in I\}.$$

Now, it can be shown that the least Herbrand model of a logic program $P$ is the least fixpoint of $T_P$. That is, let $I_0 = \emptyset$ and $I_{n+1} = T_P(I_n)$ for $n = 0, 1, 2, \ldots$, then the least Herbrand model of $P$ is the $I_m$ with smallest $m \geq 0$ such that $I_{m+1} = I_m$. In case of functor-free, range-restricted clauses, the least Herbrand model can be obtained using the following procedure:

1: Initialize LH := $\emptyset$
2: **repeat**
3: LH := $T_P$(LH)
4: **until** LH does not change anymore

That is, initialize LH to the empty set, and then add all ground facts head$(c)\theta$ to LH for which there exists a clause $c \in P$ and a substitution such that body$(c)\theta \subseteq$ LH. Such ground facts are called **immediate consequences** of body$(c)\theta$. Repeat this last step until a fixpoint [4] is reached (i.e.LH does not change any more).

**Example 2.2** At this point, the reader may want to verify that LH(*grandparent*) = {parent(jef, paul), parent(paul, ann), grandparent(jef, ann)} and LH(*nat*) = hb(*nat*). ∘

All ground atoms in the least Herbrand model are provable. **Proofs** are typically constructed using the **SLD-resolution** procedure: given a **goal** :-$\mathtt{G_1}, \mathtt{G_2} \ldots, \mathtt{G_n}$ and a clause G:-$\mathtt{L_1}, \ldots, \mathtt{L_m}$ such that $\mathtt{G_1}\theta = \mathtt{G}\theta$, applying SLD resolution yields the new goal :-$\mathtt{L_1}\theta, \ldots, \mathtt{L_m}\theta, \mathtt{G_2}\theta \ldots, \mathtt{G_n}\theta$ . A *successful* refutation, i.e., a proof of a goal is then a sequence of resolution steps yielding the empty goal, i.e. :- . *Failed* proofs do not end in the empty goal.

**Example 2.3** The atom grandparent(jeff, ann) is true because of

:-grandparent(jeff, ann)
:-parent(jeff, Z), parent(Z, ann)
:-parent(paul, ann)
:-

∘

Resolution is employed by many theorem provers (such as Prolog). Indeed, when given the goal grandparent(jeff, ann), Prolog would compute the above successful resolution refutation and answer that the goal is true.

For a detailed introduction to logic programming, we refer to [Lloyd, 1989], for a more gentle introduction, we refer to [Flach, 1994], and for a detailed discussion of Prolog, see [Sterling and Shapiro, 1986].

[4] For definite clause programs, this fixpoint always exist, is unique, and is the least Herbrand model.

## 2.2 Inductive Logic Programming (ILP) and its Settings

Inductive logic programming is concerned with finding a hypothesis $H$ (a logic program, i.e. a definite clause program) from a set of positive and negative examples *Pos* and *Neg*.

**Example 2.4** (Adapted from Example 1.1 in [Lavrač and Džeroski, 1994]) Consider learning a definition for the `daughter`/2 predicate, i.e., a set of clauses with head predicates over `daughter`/2, given the following facts as learning examples

*Pos* `daughter(dorothy, ann).`
`daughter(dorothy, brian).`
*Neg* `daughter(rex, ann).`
`daughter(rex, brian).`

Additionally, we have some general knowledge called background knowledge $B$, which describes the family relationships and sex of each person:

```
mother(ann, dorothy). female(dorothy).        female(ann).
mother(ann, rex).     father(brian, dorothy). father(brian, rex).
```

From this information, we could induce $H$

```
daughter(C, P) :- female(C), mother(P, C).
daughter(C, P) :- female(C), father(P, C).
```

which perfectly explains the examples in terms of the background knowledge, i.e., *Pos* are entailed by $H$ together with $B$, but *Neg* are not entailed. ∘

More formally, ILP is concerned with the following learning problem.

**Definition 2.5** (ILP Learning Problem) **Given** a set of positive and negative examples *Pos* and *Neg* over some language $\mathcal{L}_E$, a background theory $B$, in the form of a set of definite clauses, a hypothesis language $\mathcal{L}_H$, which specifies the clauses that are allowed in hypotheses, and a *covers* relation $covers(e, H, B) \in \{0, 1\}$, which basically returns the classification of an example $e$ with respect to $H$ and $B$, **find** a hypothesis $H$ in $\mathcal{H}$ that covers (with respect to the background theory $B$) all positive examples in *Pos* (completeness) and none of the negative examples in *Neg* (consistency). ∘

The language $\mathcal{L}_E$ chosen for representing the examples together with the *covers* relation determines the inductive logic programming setting De Raedt [1997]. Various settings have been considered in the literature [De Raedt, 1997]. In the following, we will formalize learning from *entailment* [Plotkin, 1970] and from *interpretations* [Helft, 1989, De Raedt and Džeroski, 1994]. We further introduce a novel, intermediate setting, which we call learning from *proofs*. It is inspired on the seminal work by Shapiro [1983].

### 2.2.1 Learning from Entailment

Learning from entailment is by far the most popular ILP setting and it is addressed by a wide variety of well-known ILP systems such as FOIL [Quinlan and Cameron-Jones, 1995], PROGOL [Muggleton, 1995], and ALEPH [Srinivasan, 1999].

**Definition 2.6** (Covers Relation for Learning from Entailment) When learning from entailment, the examples are definite clauses and a hypothesis $H$ covers an example $e$ with respect to the background theory $B$ if and only if $B \cup H \models e$, i.e., each model of $B \cup H$ is also a model of $e$. ◦

In many well-known systems, such as FOIL, one requires that the examples are ground facts, a special form of clauses. To illustrate the above setting, consider the following example inspired on the well-known mutagenicity application [Srinivasan et al., 1996].

**Example 2.7** Consider the following facts in the background theory $B$, which describe part of molecule 225.

| | |
|---|---|
| $\texttt{molecule}(225).$ | $\texttt{bond}(225, \texttt{f1\_1}, \texttt{f1\_2}, 7).$ |
| $\texttt{logmutag}(225, 0.64).$ | $\texttt{bond}(225, \texttt{f1\_2}, \texttt{f1\_3}, 7).$ |
| $\texttt{lumo}(225, -1.785).$ | $\texttt{bond}(225, \texttt{f1\_3}, \texttt{f1\_4}, 7).$ |
| $\texttt{logp}(225, 1.01).$ | $\texttt{bond}(225, \texttt{f1\_4}, \texttt{f1\_5}, 7).$ |
| $\texttt{nitro}(225, [\texttt{f1\_4}, \texttt{f1\_8}, \texttt{f1\_10}, \texttt{f1\_9}]).$ | $\texttt{bond}(225, \texttt{f1\_5}, \texttt{f1\_1}, 7).$ |
| $\texttt{atom}(225, \texttt{f1\_1}, \texttt{c}, 21, 0.187).$ | $\texttt{bond}(225, \texttt{f1\_8}, \texttt{f1\_9}, 2).$ |
| $\texttt{atom}(225, \texttt{f1\_2}, \texttt{c}, 21, -0.143).$ | $\texttt{bond}(225, \texttt{f1\_8}, \texttt{f1\_10}, 2).$ |
| $\texttt{atom}(225, \texttt{f1\_3}, \texttt{c}, 21, -0.143).$ | $\texttt{bond}(225, \texttt{f1\_1}, \texttt{f1\_11}, 1).$ |
| $\texttt{atom}(225, \texttt{f1\_4}, \texttt{c}, 21, -0.013).$ | $\texttt{bond}(225, \texttt{f1\_11}, \texttt{f1\_12}, 2).$ |
| $\texttt{atom}(225, \texttt{f1\_5}, \texttt{o}, 52, -0.043).$ | $\texttt{bond}(225, \texttt{f1\_11}, \texttt{f1\_13}, 1).$ |

...

$\texttt{ring\_size\_5}(225, [\texttt{f1\_5}, \texttt{f1\_1}, \texttt{f1\_2}, \texttt{f1\_3}, \texttt{f1\_4}]).$

$\texttt{hetero\_aromatic\_5\_ring}(225, [\texttt{f1\_5}, \texttt{f1\_1}, \texttt{f1\_2}, \texttt{f1\_3}, \texttt{f1\_4}]).$

...

Consider now the positive example $\texttt{mutagenic}(225)$. It is covered by $H$

$$\texttt{mutagenic}(\texttt{M}) \ :- \ \texttt{nitro}(\texttt{M}, \texttt{R1}), \texttt{logp}(\texttt{M}, \texttt{C}), \texttt{C} > 1.$$

together with the background knowledge $B$, because $H \cup B$ entails the example. To see this, we unify $\texttt{mutagenic}(225)$ with the clause's head. This yields $\texttt{mutagenic}(225) \ :- \ \texttt{nitro}(225, \texttt{R1}), \texttt{logp}(225, \texttt{C}), \texttt{C} > 1$. Now, $\texttt{nitro}(225, \texttt{R1})$ unifies with the fifth ground atom (left-hand side column) in $B$, and $\texttt{logp}(225, \texttt{C})$ with the fourth one. Because $1.01 > 1$, we found a proof of $\texttt{mutagenic}(225)$. ◦

### 2.2.2 Learning from Interpretations

The learning from interpretations setting [De Raedt and Džeroski, 1994] upgrades boolean concept-learning in computational learning theory [Valiant, 1984].

**Definition 2.8** (Covers Relational for Learning from Interpretations) When learning from interpretations, the examples are Herbrand interpretations and a hypothesis $H$ covers an example $e$ with respect to the background theory $B$ if and only if $e$ is a model of $B \cup H$. ∘

Recall that Herbrand interpretations are sets of true ground facts and they completely describe a possible situation.

**Example 2.9** Consider the interpretation $I$, which is the union of $B$

$$\begin{aligned} B = \{&\texttt{father}(\texttt{henry},\texttt{bill}), \texttt{father}(\texttt{alan},\texttt{betsy}), \texttt{father}(\texttt{alan},\texttt{benny}),\\ &\texttt{father}(\texttt{brian},\texttt{bonnie}), \texttt{father}(\texttt{bill},\texttt{carl}), \texttt{father}(\texttt{benny},\texttt{cecily}),\\ &\texttt{father}(\texttt{carl},\texttt{dennis}), \texttt{mother}(\texttt{ann},\texttt{bill}), \texttt{mother}(\texttt{ann},\texttt{betsy}),\\ &\texttt{mother}(\texttt{ann},\texttt{bonnie}), \texttt{mother}(\texttt{alice},\texttt{benny}), \texttt{mother}(\texttt{betsy},\texttt{carl}),\\ &\texttt{mother}(\texttt{bonnie},\texttt{cecily}), \texttt{mother}(\texttt{cecily},\texttt{dennis}), \texttt{founder}(\texttt{henry}),\\ &\texttt{founder}(\texttt{alan}), \texttt{founder}(\texttt{ann}), \texttt{founder}(\texttt{brian}), \texttt{founder}(\texttt{alice})\} \end{aligned}$$

and

$$C = \{\texttt{carrier}(\texttt{alan}), \texttt{carrier}(\texttt{ann}), \texttt{carrier}(\texttt{betsy})\}\ .$$

The interpretation $I$ is covered by the clause $c$

$$\texttt{carrier}(\texttt{X}) \ :- \ \texttt{mother}(\texttt{M},\texttt{X}), \texttt{carrier}(\texttt{M}), \texttt{father}(\texttt{F},\texttt{X}), \texttt{carrier}(\texttt{F}).$$

because $I$ is a model of $c$, i.e., for all substitutions $\theta$ such that $\text{body}(c)\theta \subseteq I$, it holds that $\text{head}(c)\theta \in I$. ∘

The key difference between learning from interpretations and learning from entailment is that interpretations carry much more — even complete — information. Indeed, when learning from entailment, an example can consist of a *single* fact, whereas when learning from interpretations, all facts that hold in the example are known. Therefore, learning from interpretations is typically easier and computationally more tractable than learning from entailment, cf. [De Raedt, 1997].

### 2.2.3 Learning from Proofs

Because learning from entailment (with ground facts as examples) and interpretations occupy extreme positions with respect to the information the examples carry, it is interesting to investigate intermediate positions. Ehud Shapiro's [1983] Model Inference System (MIS) fits nicely within the learning from entailment setting where examples are facts. However, to deal with missing information, Shapiro employs a clever strategy: MIS queries the users for missing information by asking them for the truth-value of facts. The answers to these queries allow MIS to reconstruct the trace or the proof of the positive examples. Inspired by Shapiro, we define the learning from proofs setting.

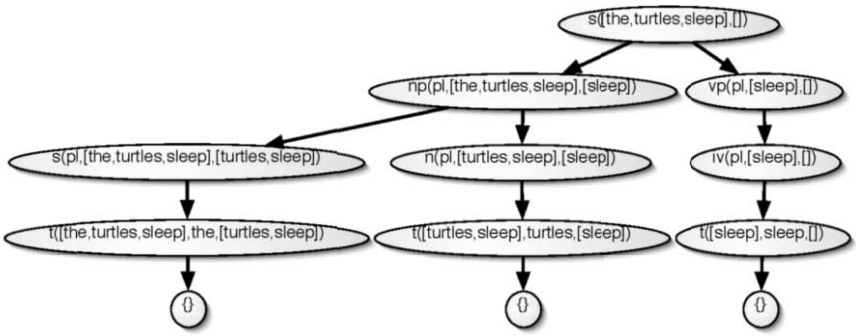



**Figure 2.2.** A proof tree, which is covered by the definite clause grammar in Example 2.12. Symbols are abbreviated.

**Definition 2.10** (Covers Relation for Learning from Proofs) When learning from proofs, the examples are ground proof-trees and an example $e$ is covered by a hypothesis $H$ with respect to the background theory $B$ if and only if $e$ is a proof-tree for $H \cup B$. ∘

At this point, there exist various possible forms of proof-trees. Here, we will — for reasons that will become clear later — assume that the proof-tree is given in the form of a ground and-tree where the nodes contain ground atoms. More formally:

**Definition 2.11** (Proof Tree) A tree $t$ is a proof-tree for a logic program $T$ if and only if $t$ is a rooted tree where for every node $n \in t$ with $children(n)$ satisfies the property that there exists a substitution $\theta$ and a clause $c \in T$ such that $n = \text{head}(c)\theta$ and $children(n) = \text{body}(c)\theta$. ∘

**Example 2.12** Consider the following *definite clause grammar*.

```
sentence(A, B) :- noun_phrase(C, A, D), verb_phrase(C, D, B).
noun_phrase(A, B, C) :- article(A, B, D), noun(A, D, C).
verb_phrase(A, B, C) :- intransitive_verb(A, B, C).
article(singular, A, B) :- terminal(A, a, B).
article(singular, A, B) :- terminal(A, the, B).
article(plural, A, B)   :- terminal(A, the, B).
noun(singular, A, B) :- terminal(A, turtle, B).
noun(plural, A, B)   :- terminal(A, turtles, B).
intransitive_verb(singular, A, B) :- terminal(A, sleeps, B).
intransitive_verb(plural, A, B) :- terminal(A, sleep, B).
terminal([A|B],A,B).
```

It covers the proof tree shown in Figure 2.2 ∘

Proof-trees contain — as interpretations — a lot of information. Indeed, they contain instances of the clauses that were used in the proofs. Therefore, it may be hard for the

user to provide this type of examples. Even though this is generally true, there exist specific situations for which this is feasible. Indeed, consider tree banks such as the UPenn Wall Street Journal corpus [Marcus et al., 1994], which contain parse trees. These trees directly correspond to the proof-trees we talk about. Another example is *explanation-based learning* (EBL) [Ellman, 1989, Mooney and Zelle, 1994]. It uses an existing domain theory to deductively explain an example **(explanation step)** in terms of a proof-tree and variablizes the explanation, i.e., generalizes the proof as far as possible while maintaining it's correctness **(generalization step)**.

### 2.2.4 Inductive Logic Programming Techniques

Given the different learning settings, there are — broadly speaking — three types of ILP approaches. One can start from short clauses, iteratively adding literals to their bodies as long as they do not become to overly general (*top-down* approaches); one can start from long clauses, iteratively removing literals until they would become overly general (*bottom-up* approaches); or, one can follow an *hybrid* approach mixing top-down and bottom-up searches. Hybrid approaches are usually employed for *multiple predicate learning* [De Raedt et al., 1993] and *theory revision* [Wrobel, 1996].

Basically, in **top-down** approaches, hypotheses are generated in a pre-determined order, and then tested against the examples. More precisely, they start with the most general hypothesis, i.e., clauses of the form $\texttt{daugther}(\texttt{C},\texttt{P}) \texttt{ :- } \texttt{true}$ where all arguments are distinct variables. After seeing the first example contradicting the hypothesis, i.e., after seeing the first negative example, the hypothesis is specialized by selecting a clause, which is then is specialized typically in three ways: by applying a substitution, by adding a literal, i.e., an atom or its negation to the body, and by adding a new clause which in turn is specialized. For instance, we can consider $\texttt{daugther}(\texttt{C},\texttt{P}) \texttt{ :- } \texttt{female}(\texttt{C})$ and $\texttt{daugther}(\texttt{C},\texttt{P}) \texttt{ :- } \texttt{mother}(\texttt{P},\texttt{C})$ for further investigations. Several possibilities (and successive specializations) have to be tried before one finds a clause that covers some positive examples but no negative ones such as $\texttt{daugther}(\texttt{C},\texttt{P}) \texttt{ :- } \texttt{female}(\texttt{C}),\texttt{mother}(\texttt{P},\texttt{C})$. Because some positive examples are still not covered, we add a new, maximally general clause to the hypothesis and essentially iterate the process as before until all positive examples are covered and no negative example. To employ the background knowledge $B$, it will be convenient — for the purpose of this thesis — to view the background knowledge $B$ as a logic program (i.e. a definite clause program) that is provided to the inductive logic programming system and fixed during the learning process. The hypothesis $H$ together with the background theory $B$ should cover all positive and none of the negative examples.

Top-down approaches are for instance often employed by ILP systems that learn from entailment. More precisely, these systems often employ a separate-and-conquer rule-learning strategy [Fürnkranz, 1999]. In an outer loop of the algorithm, they follow a set-covering approach [Mitchell, 1997] in which they repeatedly search for a rule covering many positive examples and none of the negative examples. They then delete the positive examples covered by the current clause and repeat this process until all positive examples have been covered. In the inner loop of the algorithm, they typically refine a clause by unifying variables, by instantiating variables to constants, and/or by adding literals to the clause. ILP systems that learn from interpretations

work in a similar fashion as those that learn from entailment. There is, however, one crucial difference and it concerns the generality relationship, see Definition 2.15: When **learning from entailment**, $G$ is more general than $S$ if and only if $G \models S$, whereas when **learning from interpretations**, when $S \models G$. Another difference is that learning from interpretations is well suited for learning from positive examples only. For this case, a complete search of the space ordered by $\theta$-subsumption is performed until all clauses cover all examples [De Raedt and Dehaspe, 1997].

While top-down approaches successively specialize a very general starting hypothesis, **bottom-up** approaches successively generalize a very specific hypothesis. This is basically done by deleting literals (or clauses), by turning constants into variables and/or bounded variables into new variables. Reconsider for instance the learning from proofs setting. By analogy with the learning of tree-bank grammars, one could turn all the proof-trees (corresponding to positive examples) into a set of ground clauses, which would constitute the initial theory. This theory can then be generalized by taking the least general generalization (under $\theta$-subsumption) of pairwise clauses. Of course, care must be taken that the generalized theory does not cover negative examples. For more details, we refer to Section[5] 2.4.4.

Thus, ILP approaches iteratively modify the current hypothesis syntactically and test it against the examples and background theory. The syntactic modifications are done using so-called *refinement operators* [Shapiro, 1983, Nienhuys-Cheng and de Wolf, 1997], which make small modifications to a hypothesis.

**Definition 2.13** (Refinement Operator) A refinement operator $\rho : \mathcal{H} \mapsto 2^{\mathcal{H}}$ takes an hypothesis $H \in \mathcal{H}$ and gives back a syntactically modified version $H' \in \mathcal{H}$ of $H$. ∘

For clauses, generalization and specialization operators $\rho_g$ and $\rho_s$ are usually employed, which just basically add a literal, unify variables, and ground variables respectively which delete a literal, anti-unify variables, and replace constants with variables.

**Example 2.14** Specializations of the clause $\texttt{daughter(C,P)} \texttt{ :- female(C)}$ include

| | |
|---|---|
| $\texttt{daughter(C,ann)} \texttt{ :- female(C), mother(C,ann)}$ | by grounding variables, |
| $\texttt{daughter(C,C)} \texttt{ :- female(C), mother(C,C)}$ | by unifying variables, |
| $\texttt{daughter(C,P)} \texttt{ :- female(C), mother(C,P)}$ | by adding a literal. |

Generalizations of $\texttt{daughter(C,rex)} \texttt{ :- female(C), mother(C,rex)}$ include

| | |
|---|---|
| $\texttt{daughter(C,P)} \texttt{ :- female(C), mother(C,P)}$ | by turning constants into variables, |
| $\texttt{daughter(C,P)} \texttt{ :- female(C), mother(C,P)}$ | by anti-unifying variables, |
| $\texttt{daughter(C,P)} \texttt{ :- female(C)}$ | by deleting a literal. |

∘

[5] To the best of the author's knowledge (but see [Shapiro, 1983, Bergadano and Gunetti, 1996] for ILP systems that learn from traces), no ILP system has been developed to learn from proof-trees.

Refinement operators $\rho$ on clauses can straightforwardly be extended to logic programs $H$ by defining $\rho(H) = H \setminus \{c\} \cup \{\rho(c)\}$. Based on refinement operators, ILP systems traverse the hypothesis space $\mathcal{H}$, which consists of all logic programs over $\mathcal{L}_H$, according to some generality notation.

**Definition 2.15** (More–General–Than Relation) A hypothesis $G$ *is more general than* a hypothesis $S$ if all examples covered by $S$ are also covered by $G$. ◦

Several generality frameworks have been proposed including inverse implication, inverse resolution and inverse entailment. In practice, however, the large majority of ILP systems uses Plotkin's [1970] framework of $\theta$-subsumption, see Section 2.1. This is due to better computational properties of $\theta$-subsumption. Entailment between clauses is undecidable, whereas $\theta$-subsumption is decidable (but NP-complete). The evaluation on the examples is done using the *covers* relation, which basically returns the classification $covers(e, H, B) \in \{0, 1\}$ of an example $e$ with respect to $H$ and $B$. Indeed, using the covers relation suffers from several problems such as plateaus, i.e., hypotheses with the same coverage, noise, and overfitting [6]. Thus, using the *covers* relation only is inefficient except for relatively restricted induction problems. To overcome the problem, ILP system typically resort to a heuristic function score to direct search. Several heuristics have been developed including Laplace estimates, MDL-based measures, and Bayesian approaches. In addition, ILP systems usually include a stopping criterion that is related to the significance of the heuristic score.

It should be stressed that ILP is a difficult problem. Practical ILP systems fight the inherent complexity of the problem by imposing all sorts of constraints, mostly syntactic in nature. Such constraints include *language* and *search biases*, and are sometimes summarized as *declarative* biases, see [Nédellec et al., 1996] for an overview. Essentially, the main source of complexity in ILP steams from the variables in the clauses. In top-down systems, the branching factor of the specialization operator increases with the number of variables in the clauses. Introducing *types* for predicates can rule out main potential substitutions and unifications.

**Example 2.16** The type definition $\texttt{type}(\texttt{father}(\texttt{person}, \texttt{person}))$ specifies that both argument of atoms over $\texttt{father}/2$ have to be persons. ◦

Furthermore, one can put a bound in the number of distinct variables that can occur in clauses. *Mode declarations* are another well-known ILP devise. They are used to describe input-output behaviour of predicate definitions.

**Example 2.17** We might specify $\texttt{mode}(\texttt{daugther}(+, -))$ and $\texttt{mode}(\texttt{father}(-, +))$, meaning that the + arguments must be instantiated, whereas the − arguments will be bounded to the answer. ◦

[6] In general, a model can suffer from either *underfitting* or *overfitting*. A model that is not sufficiently complex can fail to fully detect the underlying rule of a complicated data set, leading to underfitting. A model that is too complex may fit the noise, not just the underlying rule, leading to overfitting and, for instance, wild predictions.

*Refinement operators* can also be used to encode a language bias, since they can be restricted to generate only a subset of the language $\mathcal{L}_H$. For instance, refinement operators can easily be modified to generate only constant-free and function-free clauses. Other methods use a kind of grammar construction to explicitly declare the range of acceptable clauses, see e.g. Cohen [1994]. *Lookaheads* are an example of a search bias. In some cases, an atom might never be chosen by our algorithm because it will not — in itself — result in a better score. However, such an atom, while not useful in itself, might introduce new variables that make a better coverage possible by adding another atoms later on [Quinlan, 1991] [7]. It is usually solved by allowing the algorithm to *look ahead* in the search space. Instead of considering refinements with a single atom, one considers larger refinements consisting of multiple atoms [Blockeel and De Raedt, 1997].

## 2.3 Probabilistic ILP Settings

Let us now extend the inductive logic programming settings to the probabilistic case. When working with probabilistic ILP representations, there are essentially two changes:

(1) clauses in $H$ and $B$ are annotated with probabilistic information, and
(2) the covers relation becomes probabilistic.

A probabilistic covers relation softens the hard covers relation employed in traditional ILP and is defined as the probability of an example given the hypothesis and the background theory.

**Definition 2.18** (Probabilistic Covers Relation) A probabilistic covers relation takes as arguments an example $e$, a hypothesis $H$ and possibly the background theory $B$, and returns the probability value $\mathbf{P}(e \mid H, B)$ between 0 and 1 of the example $e$ given $H$ and $B$, i.e., $\text{covers}(e, H, B) = \mathbf{P}(e \mid H, B)$ . ∘

Here, we use the following probability notations. With $\mathtt{x}$, we denote a (random) variable. Furthermore, $x$ denotes a state and $\mathbf{X}$ (resp. $\boldsymbol{x}$) a set of variables (resp. states). We will use $\mathbf{P}$ to denote a probability distribution, e.g., $\mathbf{P}(\mathtt{x})$, and $P$ to denote a probability value, e.g., $P(\mathtt{x} = x)$ and $P(\mathbf{X} = \boldsymbol{x})$ .

Using the probabilistic covers relation of Definition 2.18, our first attempt at a definition of the probabilistic ILP learning problem is as follows.

**Preliminary Definition 2.19** (Probabilistic ILP Learning Problem) **Given** a probabilistic-logical language $\mathcal{L}_H$ and a set $E$ of examples over some language $\mathcal{L}_E$, **find** the hypothesis $H^*$ in $\mathcal{L}_H$ that maximizes $\mathbf{P}(E \mid H^*, B)$. ∘

Under the usual **i.i.d.** assumption, i.e., examples are sampled independently from identical distributions, this results in the maximization of

$$\mathbf{P}(E \mid H^*, B) = \prod_{e \in E} \mathbf{P}(e \mid H^*, B) = \prod_{e \in E} \text{covers}(e, H^*, B) \; .$$

[7] This effect has also been observed when learning Bayesian networks [Xiang et al., 1996].

Similar to the ILP learning problem, the language $\mathcal{L}_E$ selected for representing the examples together with the probabilistic covers relation determines different learning setting. Guided by Definition 2.19, we will now introduce three probabilistic ILP settings, which extend the purely logical ones sketched before. Afterwards, we will refine Definition 2.19 in Definition 2.26.

### 2.3.1 Probabilistic Learning from Interpretations

In order to integrate probabilities in the learning from interpretations setting, we need to find a way to assign probabilities to interpretations covered by an annotated logic program. In the past few years, this issue has received a lot of attention and various different approaches have been developed such as probabilistic-logic programs [Ngo and Haddawy, 1997], probabilistic relational models [Pfeffer, 2000], relational Baysian networks Jäger [1997], and Bayesian logic programs [Kersting, 2000, Kersting and De Raedt, 2001b]. Here, we focus on Domingos and Richardson's [2004] Markov logic networks (MLNs) as the probabilistic ILP system. Bayesian logic programs will be discussed in detail in Part I.

Markov logic networks combine Markov networks [Pearl, 1991], which represent probability distributions over propositional interpretations, with first order logic. The idea underlying Markov logic networks is to view logical formulas as soft constraints on the set of possible worlds, i.e., interpretations: if a world violates one formula, it is less probable but not necessarily impossible as in classical logic. The fewer formulas a world violates, the more probable it is. In a Markov logic network, this is realized by associating a weight with each formula that reflects how strong the constraint is. More precisely, a Markov logic network consists of weighted first-order predicate logic formulae $H = \{C_1, C_2, \ldots, C_m\}$. The weights $w_C$ of a formula $C$ specify a bias for ground instances to be true in a logical model. Consider the following example taken from [Richardson and Domingos, 2005].

**Example 2.20** *Friends-smokers* is a small Markov logic network that calculates the probability of a person $\mathtt{P}$ having lung cancer $\mathtt{ca(P)}$ based whether or not a person or her friends $\mathtt{fr(P,P')}$ smokes $\mathtt{sm(P)}$ respectively $\mathtt{sm(P')}$. This can be encoded using the following Markov logic formulas:

$1.5: \ \forall \mathtt{X} \ : \ \mathtt{sm(X)} \Rightarrow \mathtt{ca(X)}$

$1.1: \ \forall \mathtt{X,Y} \ : \ \mathtt{fr(x;y)} \Rightarrow (\mathtt{sm(X)} \Leftrightarrow \mathtt{sm(Y)})$

∘

For a given finite domains (roughly speaking a finite set of constants) $D = \{d_1, d_2, \ldots, d_n\}$, the Markov logic network defines a probability distribution over interpretations $I$ over domain $D$ and the relations occurring in the Markov logic network via

$$P(I|H,B) = \frac{1}{Z(I)} \prod_{C \in H \cup B} e^{n_C(I) \cdot w_C} = \frac{1}{Z(I)} \prod_{C \in H \cup B} \phi_C(I)^{n_C(I)} \tag{2.1}$$

where $n_C(I)$ is the number of true groundings of $C$ in $I$, $\phi_C(I) = e^{w_C}$, and $B$ is a possible background theory.

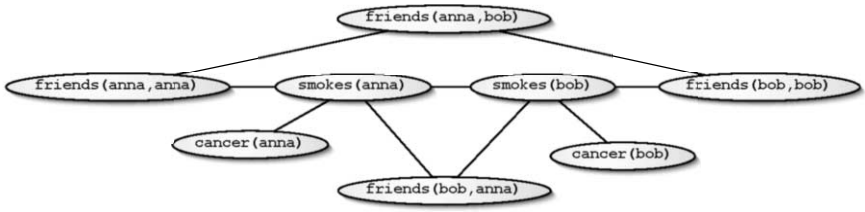



**Figure 2.3.** The Markov network induced by the *friends-smoker* Markov logic network assuming `anna` and `bob` as constants.

Markov logic networks can be viewed as proving templates for constructing Markov networks. Given a set $D$ constants, the nodes correspond to the ground atoms in the Herbrand base of the corresponding set of formulas $C$ and there is an edge between two nodes if and only if the corresponding ground atoms appear together in at least one grounding of one formula $C_i$.

**Example 2.21** Assuming `anna` and `bob` as constants, the *friends-smoker* Markov logic network induces the Markov network in Figure 2.3. ○

Note that given different sets of constants, the Markov logic network will produce different Markov networks. From Equation (2.1), we can see that an example $e$ consists of a **logical part**, which is a Herbrand interpretation of the annotated logic program, and a **probabilistic part**, which is a partial state assignment of the random variables occurring in the logical part. To see this, consider the following example.

**Example 2.22** A possible example $I$ in the *friends-smokers* domain is

$$\{\texttt{friends}(\texttt{anna},\texttt{bob}) = true,\ \texttt{friends}(\texttt{bob},\texttt{anna}) = true,\ \texttt{friends}(\texttt{anna},\texttt{anna}) = ?,$$
$$\texttt{friends}(\texttt{bob},\texttt{bob}) = true,\quad \texttt{smokes}(\texttt{anna}) = false,\quad \texttt{smokes}(\texttt{bob}) = ?,$$
$$\texttt{cancer}(\texttt{anna}) = ?,\quad \texttt{cancer}(\texttt{bob}) = false\}$$

where ? denotes an unobserved state. ○

The covers relation for $e$ can now be computed using any Markov network inference engine based on Equation (2.1).

### 2.3.2 Probabilistic Proofs

To define probabilities on proofs, ICL [Poole, 1993], PRISMs [Sato, 1995, Sato and Kameya, 2001], and stochastic logic programs [Eisele, 1994, Muggleton, 1996, Cussens, 2001] attach probabilities to facts (respectively clauses) and treat them as stochastic choices within resolution. Relational Markov models [Anderson et al., 2002] and logical hidden Markov models, which we will introduce in Part II, can be viewed as a simple fragment of them, where heads and bodies of clauses are singletons only, so-called iterative clauses. We will illustrate probabilistic learning from proofs using stochastic logic programs. For a discussion of the close relationship among stochastic logic programs, ICL, and PRISM, we refer to [Cussens, 2005].

Stochastic logic programs are inspired on stochastic context free grammars [Abney, 1997, Manning and Schütze, 1999]. The analogy between context free grammars and logic programs is that

- grammar rules correspond to definite clauses,
- sentences (or strings) to atoms, and
- productions to derivations.

Furthermore, in stochastic context-free grammars, the rules are annotated with probability labels in such a way that the sum of the probabilities associated to the rules defining a non-terminal is 1.0 .

Eisele and Muggleton have exploited this analogy to define stochastic logic programs. These are essentially definite clause programs, where each clause $c$ has an associated probability label $p_c$ such that the sum of the probabilities associated to the rules defining any predicate is 1.0 (though Cussens [1999] considered less restricted versions as well).

This framework allows ones to assign probabilities to proofs for a given predicate $q$ given a stochastic logic program $H \cup B$ in the following manner. Let $D_{\mathtt{q}}$ denote the set of all possible ground proofs for atoms over the predicate $\mathtt{q}$. For simplicity reasons, it will be assumed that there is a finite number of such proofs and that all proofs are finite (but again see [Cussens, 1999] for the more general case). Now associate to each proof $t_q \in D_q$ the probability

$$v_t = \prod_c p_c^{n_{c,t}}$$

where the product ranges over all clauses $c$ and $n_{c,t}$ denotes the number of times clause $c$ has been used in the proof $t_q$. For stochastic context free grammars, the values $v_t$ correspond to the probabilities of the production. However, the difference between context free grammars and logic programs is that in grammars two rules of the form $\mathtt{n} \rightarrow \mathtt{q}, \mathtt{n}_1, ..., \mathtt{n}_\mathtt{m}$ and $\mathtt{q} \rightarrow \mathtt{q}_1, ..., \mathtt{q}_\mathtt{k}$ always 'resolve' to give $\mathtt{n} \rightarrow \mathtt{q}_1, ..., \mathtt{q}_\mathtt{k}, \mathtt{n}_1, ..., \mathtt{n}_\mathtt{m}$ whereas resolution may fail due to unification. Therefore, the probability of a proof tree $t$ in $D_q$, i.e., a successful derivation is

$$P(t \mid H, B) = \frac{v_t}{\sum_{s \in D_q} v_s} \; . \tag{2.2}$$

The probability of a ground atom $\mathtt{a}$ is then defined as the sum of all the probabilities of all the proofs for that ground atom.

$$P(\mathtt{a} \mid H, B) = \sum_{\substack{s \in D_q \\ s \text{ is a proof for } a}} v_s \; . \tag{2.3}$$

**Example 2.23** Consider a stochastic variant of the definite clause grammar in Example 2.12 with uniform probability values for each predicate. The value $v_u$ of the proof (tree) $u$ in Example 2.12 is $v_u = \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{12}$ . The only other ground proofs $s_1, s_2$ of atoms over the predicate `sentence` are those of

$$\mathtt{sentence}([\mathtt{a}, \mathtt{turtle}, \mathtt{sleeps}], [])$$
$$\text{and } \mathtt{sentence}([\mathtt{the}, \mathtt{turtle}, \mathtt{sleeps}], []) \; .$$

Both get the value $v_{s_1} = v_{s_2} = \frac{1}{12}$. Because there is only one proof for each of the sentences,

$$P(\texttt{sentence}([\texttt{the}, \texttt{turtles}, \texttt{sleep}], [])) = v_u = \frac{1}{3} \, .$$

○

For stochastic logic programs, there are at least two natural learning settings.

Motivated by Equation (2.2), we can learn them from proofs. This makes structure learning for stochastic logic programs relatively easy, because proofs carry a lot information about the structure of the underlying stochastic logic program. Furthermore, the learning setting can be considered as an extension of the work on learning stochastic grammars from proof-banks. It should therefore also be applicable to learning unification based grammars. We will present a probabilistic ILP approach within the learning from proofs setting in Section 2.4.4.

On the other hand, we can use Equation (2.3) as covers relation and, hence, employ the learning from entailment setting. Here, the examples are ground atoms entailed by the target stochastic logic program. Learning stochastic logic programs from atoms only is much harder than learning them from proofs because atoms carry much less information than proofs. Nevertheless, this setting has been studied by Cussens [2001] and by Sato and Kameya [2001], who solves the parameter estimation problem for stochastic logic programs respectively PRISM programs, and by Muggleton [2000a, 2002], who presents an approach to structure learning of stochastic logic programs: adding one clause at a time to an existing stochastic logic program. In the following section, we will introduce the probabilistic learning from entailment. Instead of considering stochastic logic programs, however, we will study a Naïve Bayes framework, which has a much lower computational complexity.

### 2.3.3 Probabilistic Learning from Entailment

In order to integrate probabilities in the entailment setting, we need to find a way to assign probabilities to clauses that are entailed by an annotated logic program. Since most ILP systems working under entailment employ ground facts for a single predicate as examples, and the authors are unaware of any existing probabilistic ILP formalisms that implement a probabilistic covers relation for definite clauses as examples in general, we will restrict our attention to assign probabilities to facts for a single predicate. It remains an open question as how to formulate more general frameworks for working with entailment.

More formally, let us annotate a logic program $H$ consisting of a set of clauses of the form $p \leftarrow b_i$, where $p$ is an atom of the form $p(V_1, ..., V_n)$ with the $V_i$ different variables, and the $b_i$ are different bodies of clauses. Furthermore, we associate to each clause in $H$ the probability values $\mathbf{P}(b_i \mid p)$; they constitute the conditional probability distribution that for a random substitution $\theta$ for which $p\theta$ is ground and true (resp. false), the query $b_i\theta$ succeeds (resp. fails) in the knowledge base $B$. [8] Furthermore, we assume the prior probability of $p$ is given as $\mathbf{P}(p)$, it denotes the probability that

[8] The query $q$ succeeds in $B$ if there is a substitution $\sigma$ such that $B \models q\sigma$.

for a random substitution $\theta$, $p\theta$ is true (resp. false). This can then be used to define the covers relation $\mathbf{P}(p\theta \mid H, B)$ as follows (we delete the $B$ as it is fixed):

$$\mathbf{P}(p\theta \mid H) = \mathbf{P}(p\theta \mid b_1\theta, ..., b_k\theta) = \frac{\mathbf{P}(b_1\theta, ..., b_k\theta \mid p\theta) \times \mathbf{P}(p\theta)}{\mathbf{P}(b_1\theta, ..., b_k\theta)} \tag{2.4}$$

For instance, applying the naïve Bayes assumption yields

$$\mathbf{P}(p\theta \mid H) = \frac{\prod_i \mathbf{P}(b_i\theta \mid p\theta) \times \mathbf{P}(p\theta)}{\mathbf{P}(b_1\theta, ..., b_k\theta)} \tag{2.5}$$

Finally, since $P(p\theta \mid H) + P(\neg p\theta \mid H) = 1$ , we can compute $P(p\theta \mid H)$ without $P(b_1\theta, ..., b_k\theta)$ through normalization.

**Example 2.24** Consider again the mutagenicity domain and the following annotated logic program:

$(0.01, 0.21)$ : $\texttt{mutagenetic(M)} \leftarrow \texttt{atom(M, \_, \_, 8, \_)}$

$(0.38, 0.99)$ : $\texttt{mutagenetic(M)} \leftarrow \texttt{bond(M, \_, A, 1), atom(M, A, c, 22, \_), bond(M, A, \_, 2)}$

We denote the first clause by $\texttt{b}_1$ and the second one by $\texttt{b}_2$. The vectors on the left-hand side of the clauses specify $P(b_i\theta = true \mid p\theta = true)$ and $P(b_i\theta = true \mid p\theta = false)$ respectively. The covers relation (assuming the Naïve Bayes assumption) assigns probability 0.97 to example 225 because both features fail for $\theta = \{\texttt{M} \leftarrow \texttt{225}\}$. Hence,

$$\begin{aligned} P(\ \texttt{mutagenetic(225)} = true, &\texttt{b}_1\theta = false,\ \texttt{b}_2\theta = false) \\ &= P(\ \texttt{b}_1\theta = false \mid \texttt{mutagenetic(225)} = true\ ) \\ &\quad \cdot P(\ \texttt{b}_2\theta = false \mid \texttt{mutagenetic(225)} = true\ ) \\ &\quad \cdot P(\ \texttt{mutagenetic(225)} = true\ ) \\ &= 0.99 \cdot 0.62 \cdot 0.31 \approx 0.19 \end{aligned}$$

and $P(\ \texttt{mutagenetic(225)} = false, \texttt{b}_1\theta = false,\ \texttt{b}_2\theta = false) = 0.79 \cdot 0.01 \cdot 0.68 \approx 0.005$ . This yields

$$P(\ \texttt{muta(225)} = true \mid \texttt{b}_1\theta = false,\ \texttt{b}_2\theta = false\}) = \frac{0.19}{0.19 + 0.005} \approx 0.97\ .$$

∘

## 2.4 Probabilistic ILP: A Definition and Example Algorithms

Guided by Definition 2.19, we have introduced several probabilistic ILP settings for statistical relational learning. The main idea was to lift traditional ILP settings by associating probabilistic information with clauses and interpretations and by replacing ILP's deterministic covers relation by a probabilistic one. In the discussion, we made one trivial but important observation:

**Observation** Derivations might fail.

The probability of a failure is zero and, consequently, failures are never observable. Only succeeding derivations are observable, i.e., the probabilities of such derivations are greater zero. As an extreme case, recall the negative examples *Neg* employed in the ILP learning problem definition 2.2. They are supposed to be not covered, i.e., $P(Neg|H, B) = 0$ .

**Example 2.25** Reconsider Example 2.4. `Rex` is a male person; he cannot be the `daughter` of `ann`. Thus, `daughter(rex, ann)` was listed as a negative example. ∘

Negative examples conflict with the usual view on learning examples in statistical learning. In statistical learning, we seek to find that hypothesis $H^*$, which is most likely given the learning examples:

$$H^* = \arg\max_H P(H|E) = \arg\max_H \frac{P(E|H) \cdot P(F)}{P(E)} \quad \text{with} \quad P(E) > 0 \; .$$

Thus, examples $E$ are observable, i.e., $P(E) > 0$. Therefore, we refine the preliminary probabilistic ILP learning problem definition 2.19. In contrast to the purely logical case of ILP, we do not speak of *positive* and *negative* examples anymore but of *possible* and *impossible* ones.

**Definition 2.26** (Probabilistic ILP Problem) **Given** a set $E = E_p \cup E_i$ of *possible* and *impossible* examples $E_p$ and $E_i$ (with $E_p \cap E_i = \emptyset$) over some example language $\mathcal{L}_E$, a probabilistic covers relation $\text{covers}(e, H, B) = P(e \mid H, B)$, a probabilistic logical language $\mathcal{L}_H$ for hypotheses, and a background theory $B$, **find** a hypothesis $H^*$ in $\mathcal{L}_H$ such that $H^* = \arg\max_H \text{score}(E, H, B)$ and the following constraints hold: $\forall\, e_p \in E_p \; : \; \text{covers}(e_p, H^*, B) > 0$ and $\forall\, e_i \in E_i \; : \; \text{covers}(e_i, H^*, B) = 0$. The scoring function is some objective score, usually involving the probabilistic covers relation of the possible examples such as the observed likelihood $\prod_{e_p \in E_p} \text{covers}(e_p, H^*, B)$ or some penalized variant thereof. ∘

The probabilistic ILP learning problem of Definition 2.26 unifies ILP and statistical learning in the following sense: using a deterministic covers relation (,which is either 1 or 0) yields the classical ILP learning problem, see Definition 2.2, whereas sticking to propositional logic and learning from *possible* examples, i.e., $P(E) > 0$, only yields traditional statistical learning. It furthermore makes abstraction of many particular kinds of problems. In *density estimation*, the joint probability distribution of some random variables is estimated, whereas in *classification* and *regression* the dependency of a discrete respectively continuous target variable given the value of some other variables is estimated. Furthermore, several types of learning can be distinguished. In *supervised learning*, the training examples contain information about all variables including the target variable. In *reinforcement learning*, the training examples contain only indirect target information such as the classifier did well or not. Finally, in *unsupervised learning*, no values of the target variable are observed. Another important distinction is whether all the random variables variables are observed, or whether some of them are hidden, e.g., they are specified in the background knowledge and never observed. We also formulated the learning problem as a 'point estimation' problem, i.e., the goal is to find a single best hypothesis

$H^*$, because we will focus on this setting in this thesis. In general, one could also consider *Bayesian learning*, where the goal is to return a posterior distribution over hypotheses. Finally, learning might refer to the structure, i.e., the underlying logic program of the hypothesis, the parameters, or both.

To come up with algorithms solving probabilistic ILP learning problems, say for density estimation, one typically distinguishes two subtasks because $H = (L, \lambda)$ is essentially a logic program $L$ annotated with probabilistic parameters $\lambda$:

(1) *Parameter estimation* where it is assumed that the underlying logic program $L$ is fixed, and the learning task consists of estimating the parameters $\lambda$ that maximize the likelihood.
(2) *Structure learning* where both $L$ and $\lambda$ have to be learned from the data.

Below, we will sketch basic parameter estimation and structure learning techniques, and illustrate them for each setting. In the remainder of the thesis, we will then discuss selected probabilistic ILP approaches for learning from interpretations and probabilistic learning from traces in detail. A more complete survey of learning probabilistic logic representations can be found in [De Raedt and Kersting, 2003] and in the related work sections of this thesis.

### 2.4.1 Parameter Estimation

The problem of parameter estimation is thus concerned with estimating the values of the parameters $\lambda$ of a fixed probabilistic program $H = (L, \lambda)$ that best explains the examples $E$. So, $\lambda$ is a set of parameters and can be represented as a vector. As already indicated above, to measure the extent to which a model fits the data, one usually employs the likelihood of the data, i.e. $P(E \mid L, \lambda)$, though other scores or variants could be used as well.

When all examples are fully observable, maximum likelihood reduces to frequency counting. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and all known algorithms involve nonlinear optimization The most commonly adapted technique for probabilistic logic learning is the Expectation-Maximization (EM) algorithm [Dempster et al., 1977, McLachlan and Krishnan, 1997]. EM is based on the observation that learning would be easy (i.e., correspond to frequency counting), if the values of all the random variables would be known. Therefore, it estimates these values, maximizes the likelihood based on the estimates, and then iterates. More specifically, EM assumes that the parameters have been initialized (e.g., at random) and then iteratively performs the following two steps until convergence:

**(E-Step)** On the basis of the observed data and the present parameters of the model, it computes a distribution over all possible completions of each partially observed data case.
**(M-Step)** Treating each completion as a fully observed data case weighted by its probability, it computes the improved parameter values using (weighted) frequency counting.

The frequencies over the completions are called the *expected counts*.

### 2.4.2 Structure Learning

The problem is now to learn both the structure $L$ and the parameters $\lambda$ of the probabilistic program $H = (L, \lambda)$ from data. Often, further information is given as well. As in ILP, the additional knowledge can take various different forms, including a *language bias* that imposes restrictions on the syntax of $L$, and an *initial hypothesis* $(L, \lambda)$ from which the learning process can start.

Nearly all (score-based) approaches to structure learning perform a heuristic search through the space of possible hypotheses. Typically, hill-climbing or beam-search is applied until the hypothesis satisfies the logical constraints and the score$(H, E)$ is no longer improving. The steps in the search-space are typically made using refinement operators, see Definition 2.13.

At this points, it is interesting to observe that the logical constraints often require that the possible examples are covered in the logical sense. For instance, when learning stochastic logic programs from entailment, the possible example clauses must be entailed by the logic program, and when learning Markov logic networks, the possible interpretations must be models of the underlying logic program. Thus, for a probabilistic program $H = (L_H, \lambda_H)$ and a background theory $B = (L_B, \lambda_B)$ it holds that $\forall e_p \in E_p : P(e|H, B) > 0$ if and only if covers$(e, L_H, L_B) = 1$, where $L_H$ (respectively $L_B$) is the underlying logic program (logical background theory) and covers$(e, L_H, L_B)$ is the purely logical *covers* relation, which is either 0 or 1.

Let us now sketch for each probabilistic ILP setting one learning approach.

### 2.4.3 Learning from Probabilistic Interpretations

The large majority of statistical relational learning techniques proposed so far fall into the learning from interpretations setting including parameter estimation of probabilistic logic programs [Koller and Pfeffer, 1997], learning of probabilistic relational models [Getoor et al., 2002], parameter estimation of relational Markov models Taskar et al. [2002], learning of object-oriented Bayesian networks [Bangsø et al., 2001], learning relational dependency networks [Neville and Jensen, 2004], and learning logic programs with annotated disjunctions [Vennekens et al., 2004, Riguzzi, 2004]. Also learning Bayesian logic programs, which we will address in Part I, falls into this setting. Here, we will illustrate the structure learning of clausal Markov logic networks.

Kok and Domingos [2005] proposed a beam-search based approach for learning clausal Markov logic networks from possible examples only. A *clausal* Markov logic program, see Section 2.3.1, consists of weighted clauses, i.e., disjunction of literals. The clauses without associated weights constitute a clausal program $L$, and the weights the parameters $\lambda$. Starting with some initial clausal Markov logic network $H = (L, \lambda)$, the parameters maximizing $score(L, \lambda, E)$ are computed. Then, refinement operators generalizing respectively specializing $L$ are used to to compute all neighbours of $L$ in the hypothesis space. Literals are added and deleted, and signs of literals are flipped. Each neighbour is scored, yielding new hypotheses $(L', \lambda')$. To speed-up scoring, Kok and Domingos employ a variant of the *pseudo-log-likelihood*

$$\sum_{l=1}^{n} \log P(X_l = x_l | MB_x(X_l))$$

where $x$ is the Herbrand base, $x_l$ is the $l$th ground atom's truth value, and $MB_x(X_l)$ is the state of $X_l$'s Markov blanket [9] in the data. The $b$ best ones with $score(L', \lambda', E) > score(L, \lambda, E)$ are kept. On these $b$ best ones, the refining and scoring process is iteratively applied again until no new clauses improve the score or a maximal number of literals is reached. The clause with highest score in all iterations is added to $H$, and the process is continued until no improvement in score of the current best hypothesis is obtained.

### 2.4.4 Learning from Probabilistic Proofs*

Given a training set $E$ containing ground proofs as examples, one possible approach to learning from possible proofs only combines ideas from the early ILP system Golem [Muggleton and Feng, 1992] that employs Plotkin's [1970] least general generalization (LGG) with bottom-up generalization of grammars and hidden Markov models [Stolcke and Omohundro, 1993]. The resulting algorithm employs the likelihood of the proofs $score(L, \lambda, E)$ as the scoring function. It starts by taking as $L_0$ the set of ground clauses that have been used in the proofs in the training set and scores it to obtain $\lambda_0$. After initialization, the algorithm will then repeatedly select a pair of clauses in $L_i$, and replace the pair by their LGG to yield a candidate $L'$. The candidate that scores best is then taken as $H_{i+1} = (L_{i+1}, \lambda_{i+1})$, and the process iterates until the score no longer improves. One interesting issue is that strong logical constraints can be imposed on the LGG. These logical constraints directly follow from the fact that the example proofs should still be valid proofs for the logical component $L$ of all hypotheses considered. Therefore, it makes sense to apply the LGG only to clauses that define the same predicate, that contain the same predicates, and whose (reduced) LGG also has the same length as the original clauses [10].

Preliminary results with a prototype implementation are promising. In one experiment, we generated from the target stochastic logic program

$$
\begin{aligned}
&1 : \mathsf{s}(\mathsf{A},\mathsf{B}) \leftarrow \mathsf{np}(\mathsf{Number},\mathsf{A},\mathsf{C}),\mathsf{vp}(\mathsf{Number},\mathsf{C},\mathsf{B}).\\
&1/2 : \mathsf{np}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{det}(\mathsf{A},\mathsf{C}),\mathsf{n}(\mathsf{Number},\mathsf{C},\mathsf{B}).\\
&1/2 : \mathsf{np}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{pronom}(\mathsf{Number},\mathsf{A},\mathsf{B}).\\
&1/2 : \mathsf{vp}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{v}(\mathsf{Number},\mathsf{A},\mathsf{B}).\\
&1/2 : \mathsf{vp}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{v}(\mathsf{Number},\mathsf{A},\mathsf{C}),\mathsf{np}(\mathsf{D},\mathsf{C},\mathsf{B}).\\
&1 : \mathsf{det}(\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{the},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{man},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{apple},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{men},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{apples},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{eats},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{sings},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{eat},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{sing},\mathsf{B}).
\end{aligned}
$$

[9] In a Markov network, the *Markov blanket* of a node is its set of neighbouring nodes.

* Builds on [De Raedt et al., 2005].

[10] In general, the length (number of literals) of the LGG of $m$ (ground) clauses of length at most $n$ is $n^m$, see [Muggleton and Feng, 1992].

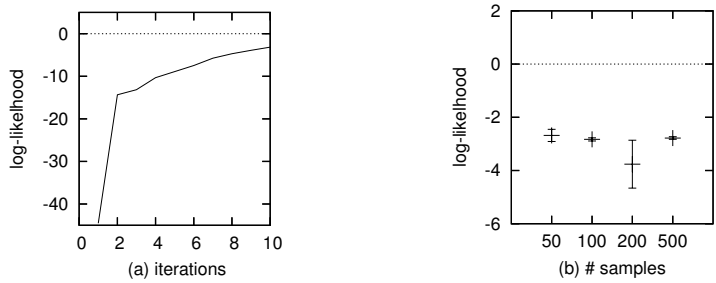



**Figure 2.4.** Experimental results on learning stochastic logic programs from proofs. **(a)** A typical learning curve. **(b)** Final log-likelihood averaged over 4 runs. The error bars show the standard deviations.

$$1 : \mathsf{pronom}(\mathsf{pl}, \mathsf{A}, \mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A}, \mathsf{you}, \mathsf{B}).$$
$$1 : \mathsf{term}([\mathsf{A}|\mathsf{B}], \mathsf{A}, \mathsf{B}) \leftarrow$$

(independent) training sets of 50, 100, 200, and 500 proofs. For each training set, 4 different random initial sets of parameters were tried. We ran the learning algorithm on each data set starting from each of the initial sets of parameters. The algorithm stopped when a limit of 200 iterations was exceeded or a change in log-likelihood between two successive iterations was smaller than 0.0001.

Figure 2.4 **(a)** shows a typical learning curve, and Figure 2.4 **(b)** summarizes the overall results. In all runs, the original structure was induced from the proof-trees. Moreover, already 50 proof-trees suffice to rediscover the structure of the original stochastic logic program. Further experiments with 20 and 10 samples respectively show that even 20 samples suffice to learn the given structure. Sampling 10 proofs, the original structure is rediscovered in one of five experiments. This supports that the *learning from proof trees* setting carries a lot information. Furthermore, our methods scales well. Runs on two independently sampled sets of 1000 training proofs yield similar results: $-4.77$ and $-3.17$, and the original structure was learned in both cases. More details can be found in [De Raedt et al., 2005].

Other statistical relational learning frameworks that have been developed within the learning from proofs setting are relational Markov models [Anderson et al., 2002] and logical hidden Markov models. The later one will be addressed in Part II.

### 2.4.5 Probabilistic Learning from Entailment*

Probabilistic learning from entailment has been investigated for learning stochastic logic programs [Muggleton, 2000a,b, Cussens, 2001, Muggleton, 2002] and for parameter estimation of PRISM programs [Sato and Kameya, 2001, Kameya et al., 2004] from possible examples only. Here, we will illustrate a promising, alternative approach with less computational complexity, which adapts FOIL [Quinlan and Cameron-Jones,

* Builds on [Landwehr et al., 2005].

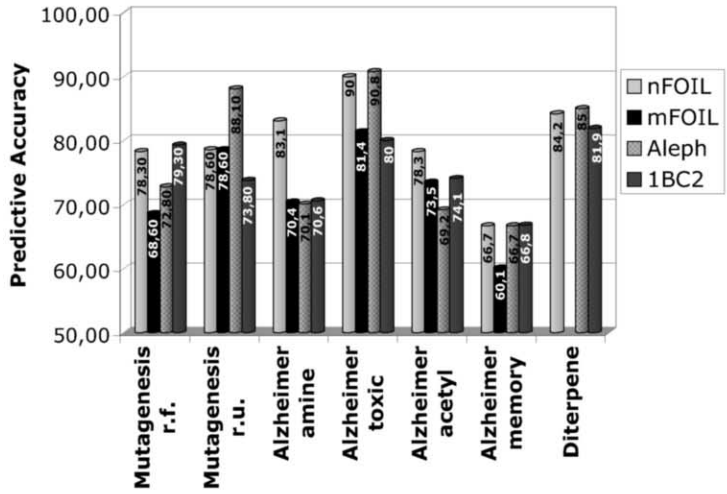



**Figure 2.5.** Cross-validated accuracy results of nFOIL on ILP benchmark data sets. For **Mutagenesis r.u.**, leave-one-out cross-validated accuracies are reported because of the small size of the data set. For all other domains, 10-fold cross-validated results are given. **mFOIL** [Lavrač and Džeroski, 1994] and **Aleph** [Srinivasan, 1999] are standard ILP algorithms. **1BC2** [Flach and Lachiche, 2004] is a first order logical variant of Naïve Bayes. For **1BC2**, we do not test significance because the results on **Mutagenesis** are taken from [Flach and Lachiche, 2004]. **Diterpene** is a multiclass problem but **mFOIL** has been developed for two-class problems only. Therefore, we do not report results for **mFOIL** on Diterpene.

1995] with the conditional likelihood as described in Equation (2.5) as the scoring function $score(L, \lambda, E)$. This idea has been followed with nFOIL, see [Landwehr et al., 2005] for more details.

Given a training set $E$ containing positive and negative examples (i.e. true and false ground facts), this algorithm stays in the learning from possible examples only to induce a probabilistic logical model to distinguish between the positive and negative examples. It computes Horn clause features $b_1, b_2, \ldots$ in an outer loop. It terminates when no further improvements in the score are obtained, i.e, when $score(\{b_1, \ldots, b_i\}, \lambda_i, E) < score(\{b_1, \ldots, b_{i+1}\}, \lambda_{i+1}, E)$, where $\lambda$ denotes the maximum likelihood parameters. A major difference with FOIL is, however, that the covered positive examples are *not* removed. The inner loop is concerned with inducing the next feature $b_{i+1}$ top-down, i.e., from general to specific. To this aim it starts with a clause with an empty body, e.g., $\texttt{muta}(\texttt{M}) \leftarrow$. This clause is then specialized by repeatedly adding atoms to the body, e.g., $\texttt{muta}(\texttt{M}) \leftarrow \texttt{bond}(\texttt{M}, \texttt{A}, 1)$, $\texttt{muta}(\texttt{M}) \leftarrow \texttt{bond}(\texttt{M}, \texttt{A}, 1), \texttt{atom}(\texttt{M}, \texttt{A}, \texttt{c}, 22, \_)$, etc. For each refinement $b'_{i+1}$ we then compute the maximum-likelihood parameters $\lambda'_{i+1}$ and $score(\{b_1, \ldots, b'_{i+1}\}, \lambda'_{i+1}, E)$. The refinement that scores best, say $b''_{i+1}$, is then considered for further refinement and the refinement process terminates when $score(\{b_1, \ldots, b_{i+1}\}, \lambda_{i+1}, E) <$

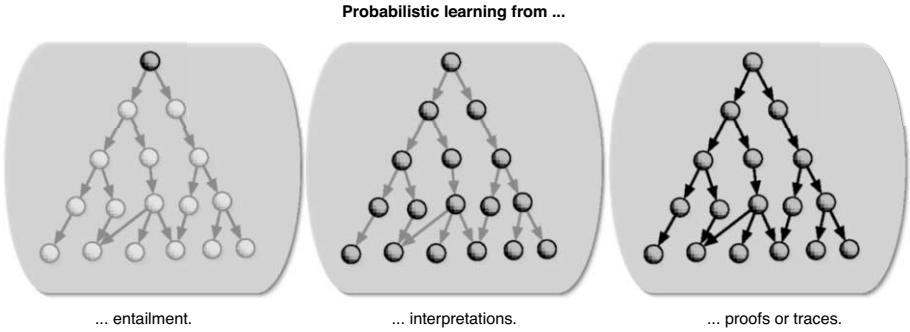



**Figure 2.6.** The level of information on the target probabilistic program provided by probabilistic ILP settings: shaded parts denote unobserved information. Learning from entailment provides the least information. Only roots of proof tree are observed. In contrast, learning from proofs or traces provides the most information. All ground clauses and atoms used in proofs are observed. Learning from interpretations provides an intermediate level of information. All ground atoms but not the clauses are observed.

$score(\{b_1, \ldots, b''_{i+1}\}, \lambda''_{i+1}, E)$. As Figure 2.5 shows, nFOIL performs well compared to other ILP systems on traditional ILP benchmark data sets. mFOIL and Aleph, two standard ILP systems, were never significantly better than nFOIL (paired sampled t-test, $p = 0.05$). nFOIL achieved significantly higher predictive accuracies than mFOIL on Alzheimer amine, toxic, and acetyl. Compared to Aleph, nFOIL achieved significantly higher accuracies on Alzheimer amine and acetyl (paired sampled t-test, $p = 0.05$). For more details, we refer to [Landwehr et al., 2005].

## 2.5 Conclusions

We have defined the formal framework of *probabilistic ILP* for statistical relational learning and presented three learning setting settings: *probabilistic learning from entailment, from interpretations*, and *from proofs*. They differ in their representation of examples and the corresponding covers relation. We have also sketched how these settings combine and generalize ILP and statistical learning.

At present, it is still an open question as to what the relation among these different settings is. It is, however, apparent that they provide different levels of information about the target probabilistic program, cf. Figure 2.6. Learning from entailment provides the least information, whereas learning from proofs or traces the most. Learning from interpretations occupies an intermediate position. This is interesting because learning is expected to be even more difficult as the less information is observed. Furthermore, the presented learning settings are by no means the only possible settings for probabilistic ILP. Examples might be weighted, e.g., by probabilities, and proofs might be partially observed only.

In the remainder of the thesis, we will go — in detail — through a number of selected probabilistic ILP frameworks, define their representation languages and semantics, design probabilistic ILP learning algorithms, and evaluate them.

This page intentionally left blank

Part I

# Probabilistic ILP over Interpretations

Bayesian networks provide an elegant formalism for representing and reasoning about uncertainty using probability theory. They are a probabilistic extension of propositional logic and, hence, inherit some of the limitations of propositional logic, such as the difficulties to represent objects and relations. Bayesian logic programs are an extension of Bayesian networks to overcome these limitations. They tightly integrate definite logic programs with Bayesian networks and, hence, define probability distributions over first-order interpretations. The key idea underlying Bayesian logic programs is to establish a one-to-one mapping between ground atoms and random variables, and between the immediate consequence operator and the dependency relation. In doing so, Bayesian logic programs combine the advantages of both definite clause logic and Bayesian networks: notions of objects and relations, a separation of quantitative and qualitative aspects of the world, and a graphical representation.

In this Part, we formally introduce *Bayesian logic programs*, their representation language, their semantics, and a graphical representation in Chapter 3. Afterwards, in Chapter 4, the structure selection problem for Bayesian logic programs is addressed. More precisely, the learning from interpretations setting from inductive logic programming is combined with score-based techniques for learning Bayesian networks and the problem of parameter estimation is reduced to the corresponding problem of (dynamic) Bayesian networks. We also briefly describe Balios, the engine for Bayesian logic programs.

This page intentionally left blank

§ 3

# Bayesian Logic Programs *

*... in which, after reviewing Bayesian networks, Bayesian logic programs are introduced, their semantics are defined, and several extensions of the basic framework are discussed: a graphical representation for Bayesian logic programs, effective treatment of logic atoms, and aggregate functions ....*

Bayesian networks [Pearl, 1991] are one of the most important, efficient and elegant frameworks for representing and reasoning with probabilistic models. They have been applied to many real-world problems in diagnosis, forecasting, automated vision, sensor fusion, and manufacturing control [Heckerman et al., 1995b]. A Bayesian network specifies a joint probability distribution over a finite set of random variables and consists of two components: (1) a *qualitative* or *logical* one that encodes the local influences among the random variables using a directed acyclic graph, and (2) a *quantitative* one that encodes the probability densities over these local influences. Despite these interesting properties, Bayesian networks also have a major limitation: they are essentially propositional representations.

**Example 3.1** Imagine the task of modeling the localization of genes/proteins. When using a Bayesian network, every gene is a single random variable. There is no way of formulating general probabilistic regularities among the localizations of the genes such as *the localization* `L` *of gene* `G` *is influenced by the localization* `L'` *of another gene* `G'` *that interacts with* `G`. ○

The propositional nature and limitation of Bayesian networks are similar to those of traditional attribute-value learning techniques, which have motivated work on upgrading these techniques within ILP. This in turn also explains the interest in upgrading Bayesian networks towards using first order logical representations.

*Bayesian logic programs* unify Bayesian networks with logic programming, which allows one to overcome the propositional character of Bayesian networks and the purely 'logical' nature of logic programs. From a knowledge representation point of view, Bayesian logic programs can be distinguished from alternative frameworks by having both logic programs (i.e. definite clause programs, which are sometimes called 'pure' Prolog programs) as well as Bayesian networks as an immediate special case. This is realized through the use of a small but powerful set of primitives. Indeed, the underlying idea of Bayesian logic programs is to establish a one-to-one mapping between ground atoms and random variables, and between the *immediate consequence operator* and the *direct influence* relation. Therefore, Bayesian logic programs can also handle domains involving structured terms as well as continuous random variables.

* Builds on [Kersting et al., 2000, Kersting, 2000, Kersting and De Raedt, 2005].

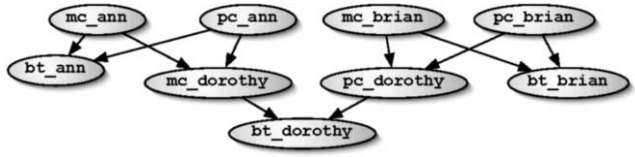



**Figure 3.1.** The graphical structure of a Bayesian network modeling the inheritance of blood types within a particular family.

## 3.1 The Propositional Case: Bayesian Networks

In this section, we first introduce the key concepts and assumptions underlying Bayesian networks. For a full and detailed treatment of Bayesian networks, we refer to [Pearl, 1991, Cowell et al., 1999, Jensen, 2001]. In the next section, we will then show how Bayesian networks and logic programs are combined in Bayesian logic programs. As running example, we will use an example from genetics, which is inspired by Friedman et al. [1999]:

**Example Domain I** (Blood Type Domain) *The blood type domain is a genetic model of the inheritance of a single gene that determines a person's* $\mathtt{X}$ *blood type* $\mathtt{bt(X)}$. *Each person* $\mathtt{X}$ *such as* `ann`, `dorothy`, *and* `brian` *has two copies of the chromosome containing this gene, one,* $\mathtt{mc(Y)}$, *inherited from her mother* $\mathtt{m(Y,X)}$, *and one,* $\mathtt{pc(Z)}$, *inherited from her father* $\mathtt{f(Z,X)}$. *Occasionally, a person is unavailable for testing, and yet because of the clarification of crime, test of paternity, allocation of (frozen) semen etc. it is often necessary that a blood type of the person be estimated. A blood type can still be derived for that person through an examination and analysis of the types of family members.* ○

A *Bayesian network* [Pearl, 1991] is an augmented, directed acyclic graph, where each node corresponds to a random variable $\mathtt{x_i}$ and each edge indicates a *direct influence* among the random variables. It represents the joint probability distribution $\mathbf{P}(\mathtt{x_1}, \ldots, \mathtt{x_n})$ over a fixed, finite set $\{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$ of random variables. Each random variable $\mathtt{x_i}$ possesses a finite set $\mathbf{S}(\mathtt{x_i})$ of mutually exclusive states.

**Example 3.2** Figure 3.1 shows the graph of a Bayesian network modeling our blood type example for a particular family. The family relationship, which is taken from Jensen's *stud farm* example [1996], forms the basis for the graph. The network encodes, e.g., that Dorothy's blood type is influenced by the genetic information of her parents Ann and Brian. The set of possible states of $\mathtt{bt(dorothy)}$ is $\mathbf{S}(\mathtt{bt(dorothy)}) = \{a, b, ab, 0\}$; the set of possible states of $\mathtt{pc(dorothy)}$ and $\mathtt{mc(dorothy)}$ are $\mathbf{S}(\mathtt{pc(dorothy)}) = \mathbf{S}(\mathtt{mc(dorothy)}) = \{a, b, 0\}$. The same holds for `ann` and `brian`. The direct predecessors of a node $\mathtt{x}$, the parents of $\mathtt{x}$ are denoted by $\mathbf{Pa}(\mathtt{x})$. For instance, $\mathbf{Pa}(\mathtt{bt(ann)}) = \{\mathtt{pc(ann)}, \mathtt{mc(ann)}\}$. ○

A Bayesian network stipulates the following conditional independence assumption.

**Independence Assumption 3.3** (Bayesian Networks) Each node $\mathtt{x_i}$ is conditionally independent of any subset $\mathbf{A}$ of nodes that are not descendants of $\mathtt{x_i}$ given a joint state of $\mathbf{Pa}(\mathtt{x_i})$, i.e. $\mathbf{P}(\mathtt{x_i} \mid \mathbf{A}, \mathbf{Pa}(\mathtt{x_i})) = \mathbf{P}(\mathtt{x_i} \mid \mathbf{Pa}(\mathtt{x_i}))$. ○

**Example 3.4** In the blood type Bayesian network of the last example, $\mathtt{bt(dorothy)}$ is conditionally independent of $\mathtt{bt(ann)}$ given a joint state of its parents $\{\mathtt{pc(dorothy)}, \mathtt{mc(dorothy)}\}$. ○

Any pair $(\mathtt{x_i}, \mathbf{Pa}(\mathtt{x_i}))$ is called the *family* of $\mathtt{x_i}$ denoted as $\mathbf{Fa}(\mathtt{x_i})$, e.g. $\mathtt{bt(dorothy)}$'s family is $(\mathtt{bt(dorothy)}, \{\mathtt{pc(dorothy)}, \mathtt{mc(dorothy)}\})$ . Because of the conditional independence assumption, we can write down the joint probability density as $\mathbf{P}(\mathtt{x_1}, \ldots, \mathtt{x_n}) = \prod_{i=1}^{n} \mathbf{P}(\mathtt{x_i} \mid \mathbf{Pa}(\mathtt{x_i}))$ by applying the independence assumption 3.3 to the chain rule expression of the joint probability distribution. Thereby, we associate with each node $\mathtt{x_i}$ of the graph the conditional probability distribution $\mathbf{P}(\mathtt{x_i} \mid \mathbf{Pa}(\mathtt{x_i}))$, denoted as $\mathrm{cpd}(\mathtt{x_i})$.

**Example 3.5** The conditional probability distributions in our blood type domain are:

| $\mathtt{mc(dorothy)}$ | $\mathtt{pc(dorothy)}$ | $\mathbf{P}(\mathtt{bt(dorothy)})$ |
|---|---|---|
| $a$ | $a$ | $(0.97, 0.01, 0.01, 0.01)$ |
| $b$ | $a$ | $(0.01, 0.01, 0.97, 0.01)$ |
| ... | ... | ... |
| 0 | 0 | $(0.01, 0.01, 0.01, 0.97)$ |

(similarly for $\mathtt{ann}$ and $\mathtt{brian}$) and

| $\mathtt{mc(ann)}$ | $\mathtt{pc(ann)}$ | $\mathbf{P}(\mathtt{mc(dorothy)})$ |
|---|---|---|
| $a$ | $a$ | $(0.98, 0.01, 0.01)$ |
| $b$ | $a$ | $(0.01, 0.98, 0.01)$ |
| ... | ... | ... |
| 0 | 0 | $(01, 0.01, 0.98)$ |

(similarly for $\mathtt{pc(dorothy)}$). Further conditional probability tables are associated with the apriori nodes, i.e., the nodes having no parents:

| $\mathbf{P}(\mathtt{mc(ann)})$ | $\mathbf{P}(\mathtt{mc(ann)})$ | $\mathbf{P}(\mathtt{mc(ann)})$ | $\mathbf{P}(\mathtt{mc(ann)})$ |
|---|---|---|---|
| $(0.38, 0.12, 0.50)$ | $(0.38, 0.12, 0.50)$ | $(0.38, 0.12, 0.50)$ | $(0.38, 0.12, 0.50)$ |

○

## 3.2 The First-Order Case

The logical component of Bayesian networks essentially corresponds to a propositional logic program. This has already been observed by Haddawy [1994] and Langley [1995]. Langley, for instance, does not represent Bayesian networks graphically but rather uses the notation of propositional definite clause programs. Consider the program in Figure 3.2. It encodes the structure of the blood type Bayesian network in Figure 3.1. Observe that the random variables in this notation correspond to logical atoms. Furthermore, the *direct influence* relation corresponds to the immediate

```
pc(ann).
pc(brian).
mc(ann).
mc(brian).
mc(dorothy) :- mc(ann), pc(ann).
pc(dorothy) :- mc(brian), pc(brian).
bt(ann) :- mc(ann), pc(ann).
bt(brian) :- mc(brian), pc(brian).
bt(dorothy) :- mc(dorothy), pc(dorothy).
```

**Figure 3.2.** A propositional clause program encoding the structure of the blood type Bayesian network in Figure 3.1.

consequence operator. Now, imagine another totally separated family, which could be described by a similar Bayesian network. The graphical structure and associated conditional probability distribution for the two families are controlled by the same intensional regularities. But these overall regularities cannot be captured by a traditional Bayesian network. So, we need another way to represent these overall regularities.

### 3.2.1 Representation Language

We upgrade the propositional clauses encoding the structure of the Bayesian network to proper first order clauses. This idea leads to the central notion of a Bayesian clause.

**Definition 3.6** (Bayesian Clause) A *Bayesian (definite) clause* $c$ *is an expression of the form* $\mathtt{A} \mid \mathtt{A}_1, \ldots, \mathtt{A}_n$ *where* $n \geq 0$*, the* $\mathtt{A}, \mathtt{A}_1, \ldots, \mathtt{A}_n$ *are Bayesian atoms (see below) and all Bayesian atoms are (implicitly) universally quantified. When* $n = 0$*,* $c$ *is called a Bayesian fact and expressed as* $\mathtt{A}$*.* ○

The differences between a *Bayesian clause* and a *logical clause* are:

(1) the atoms $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_1)$ and predicates $\mathtt{p/1}$ are Bayesian, which means that they have an associated (finite[11]) set $\mathbf{S}(\mathtt{p/1})$ of possible states, and
(2) we use '|' instead of ':−' to highlight the conditional probability distribution.

For instance, consider the Bayesian clause $c$ $\mathtt{bt(X)|mc(X), pc(X)}$ where $\mathbf{S}(\mathtt{bt/1}) = \{a, b, ab, 0\}$ and $\mathbf{S}(\mathtt{mc/1}) = \mathbf{S}(\mathtt{pc/1}) = \{a, b, 0\}$. Intuitively, a Bayesian predicate $\mathtt{p/1}$ generically represents a set of random variables. More precisely, each Bayesian ground atom $g$ over $\mathtt{p/1}$ represents a random variable over the states $\mathbf{S}(g) := \mathbf{S}(\mathtt{p/1})$. For example, $\mathtt{bt(ann)}$ represents the blood type of a person named Ann as a random variable over the states $\{a, b, ab, 0\}$. Apart from that, most *logical* notions carry over to Bayesian logic programs. So, we will speak of Bayesian predicates, terms, constants, substitutions, propositions, ground Bayesian clauses, Bayesian Herbrand interpretations etc. For the sake of simplicity we will sometimes omit the term *Bayesian* as long

[11] For the sake of simplicity we consider finite random variables, i.e. random variables having a finite set $\mathbf{S}$ of states. However, because the semantics rely on Bayesian networks, the ideas easily generalize to discrete and continuous random variables (modulo the well-known restrictions for Bayesian networks).

as no ambiguities arise. We will assume that all Bayesian clauses $c$ are range-restricted, i.e., $\mathrm{Var}(\mathrm{head}(c)) \subseteq \mathrm{Var}(\mathrm{body}(c))$. Range restriction (see 2.1) is often imposed in the database literature; it allows one to avoid the derivation of non-ground true facts. As already indicated while discussing Figure 3.2, a set of Bayesian clauses encodes the qualitative or structural component of the Bayesian logic programs. More precisely, ground atoms correspond to random variables, and the set of random variables encoded by a particular Bayesian logic program corresponds to its least Herbrand domain. In addition, the *direct influence* relation corresponds to the immediate consequence.

In order to represent a probabilistic model we also associate with each Bayesian clause $c$ a conditional probability distribution $\mathrm{cpd}(c)$ encoding $\mathbf{P}(\mathrm{head}(c) \mid \mathrm{body}(c))$, cf. Figure 3.3. To keep the exposition simple, we will assume that $\mathrm{cpd}(c)$ is represented as a table. More elaborate representations such as decision trees or rules would be possible too. The distribution $\mathrm{cpd}(c)$ generically represents the conditional probability distributions associated with each ground instance $c\theta$ of the clause $c$.

In general, one has several clauses that may even make conflicting statements on conditional probability distributions.

**Example 3.7** Consider clauses $c_1 \equiv \mathtt{bt(X)} \mid \mathtt{mc(X)}$ and $c_2 \equiv \mathtt{bt(X)} \mid \mathtt{pc(X)}$ and assume corresponding substitutions $\theta_i$ that ground the clauses $c_i$ such that $\mathrm{head}(c_1\theta_1) = \mathrm{head}(c_2\theta_2)$. In contrast to $\mathtt{bt(X)}|\mathtt{mc(X)},\mathtt{pc(X)}$, they specify $\mathrm{cpd}(c_1\theta_1)$ and $\mathrm{cpd}(c_2\theta_2)$, but not the desired distribution $\mathbf{P}(\mathrm{head}(c_1\theta_1) \mid \mathrm{body}(c_1) \cup \mathrm{body}(c_2))$. ◦

So called *combining rules* are the standard solution to obtain the distribution required.

**Definition 3.8** (Combining Rule) A combining rule is a function that maps finite sets of conditional probability distributions $\{\mathbf{P}(\mathtt{A} \mid \mathtt{A_{i1}}, \ldots, \mathtt{A_{in_i}}) \mid i = 1, \ldots, m\}$ onto one (*combined*) conditional probability distribution $\mathbf{P}(\mathtt{A} \mid \mathtt{B_1}, \ldots, \mathtt{B_k})$ with $\{\mathtt{B_1}, \ldots, \mathtt{B_k}\} \subseteq \bigcup_{i=1}^{m} \{\mathtt{A_{i1}}, \ldots, \mathtt{A_{in_i}}\}$. ◦

We assume that for each Bayesian predicate $\mathtt{p}/l$ there is a corresponding combining rule $\mathrm{cr}(\mathtt{p}/l)$, such as *noisy_or* (see e.g.[Jensen, 2001]) or *average*. The latter assumes $n_1 = \ldots = n_m$ and $\mathbf{S}(\mathtt{A_{ij}}) = \mathbf{S}(\mathtt{A_{kj}})$, and computes the average of the distributions over $\mathbf{S}(\mathtt{A})$ for each joint state over $\bigotimes_j \mathbf{S}(\mathtt{A_{ij}})$, see also the next Section 3.2.2.

By now, we are able to formally define Bayesian logic programs.

**Definition 3.9** (Bayesian Logic Program) A *Bayesian logic program* $B$ consists of a (finite) set of Bayesian clauses. For each Bayesian clause $c$ there is exactly one conditional probability distribution $\mathrm{cpd}(c)$, and for each Bayesian predicate $\mathtt{p}/l$ there is exactly one combining rule $\mathrm{cr}(\mathtt{p}/l)$. ◦

A Bayesian logic program encoding our blood type domain is shown in Figure 3.3.

### 3.2.2 Declarative Semantics

Intuitively, each Bayesian logic program represents a (possibly infinite) Bayesian network, where the nodes are the atoms in the least Herbrand model of the Bayesian logic program. These declarative semantics can be formalized using the annotated *dependency graph*.

```
m(ann, dorothy).
f(brian, dorothy).
pc(ann).
pc(brian).
mc(ann).
mc(brian).

mc(X)|m(Y, X), mc(Y), pc(Y).
pc(X)|f(Y, X), mc(Y), pc(Y).
bt(X)|mc(X), pc(X).
```

| mc(X) | pc(X) | **P**(bt(X)) |
|---|---|---|
| $a$ | $a$ | $(0.97, 0.01, 0.01, 0.01)$ |
| $b$ | $a$ | $(0.01, 0.01, 0.97, 0.01)$ |
| ... | ... | ... |
| 0 | 0 | $(0.01, 0.01, 0.01, 0.97)$ |

| m(Y, X) | mc(Y) | pc(Y) | **P**(mc(X)) |
|---|---|---|---|
| *true* | $a$ | $a$ | $(0.98, 0.01, 0.01)$ |
| *true* | $b$ | $a$ | $(0.01, 0.98, 0.01)$ |
| ... | ... | ... | ... |
| *false* | $a$ | $a$ | $(0.33, 0.33, 0.33)$ |
| ... | ... | ... | ... |

**Figure 3.3.** The *blood type* Bayesian logic program encoding our genetic domain. For each Bayesian predicate, the identity function is the combining rule. The conditional probability distributions associated with the Bayesian clauses bt(X)|mc(X), pc(X) and mc(X)|m(Y, X), mc(X), pc(Y) are represented as tables. The other distributions are correspondingly defined. The Bayesian predicates m/2 and f/2 have as possible states $\{true, false\}$.

**Definition 3.10** (Dependency Graph) The *dependency graph* $DG(B)$ is that directed graph whose nodes correspond to the ground atoms in the least Herbrand model $\mathrm{LH}(B)$. It encodes the *direct influence* relation over the random variables in $\mathrm{LH}(B)$: *there is an edge from a node* x *to a node* y *if and only if there exists a clause* $c \in B$ *and a substitution* $\theta$*, s.t.* y $=$ *head*$(c\theta)$, x $\in$ *body*$(c\theta)$ *and for all ground atoms* z *in* $c\theta$ *:* z $\in LH(B)$. ○

**Example 3.11** Figures 3.4 and 3.5 show the dependency graph for our *blood type* program. Here, mc(dorothy) *directly* influences bt(dorothy). Defining the *influence* relation as the transitive closure of the *direct influence* relation, mc(ann) influences bt(dorothy) . ○

To define the semantics of Bayesian logic programs using the dependency graph, we note that the Herbrand base $\mathrm{HB}(B)$ constitutes the set of all random variables we can talk about. However, only those atoms that are in the least Herbrand model $\mathrm{LH}(B) \subseteq \mathrm{HB}(B)$ will appear in the dependency graph. These are the atoms that are true in the logical sense, i.e., when the Bayesian logic program $B$ is interpreted as a logical program. They are the so-called *relevant* random variables, the random variables over which a probability distribution is well-defined by $B$. The atoms not belonging to the least Herbrand model are irrelevant. Now, to each node x in $DG(B)$ we associate the combined conditional probability distribution, which is the result of applying the combining rule $\mathrm{cr}(\mathtt{p}/n)$ of the corresponding Bayesian predicate $\mathtt{p}/n$ to the set of $\mathrm{cpd}(c\theta)$'s where $\mathrm{head}(c\theta) = \mathtt{x}$ and $\{\mathtt{x}\} \cup \mathrm{body}(c\theta) \subseteq \mathrm{LH}(B)$.

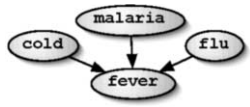
**Example 3.12** Consider the Bayesian logic program

```
cold.          fever | cold.
flu.           fever | flu.
malaria.       fever | malaria.
```

```
m(ann,dorothy).
f(brian,dorothy).
pc(ann).
pc(brian).
mc(ann).
mc(brian).
mc(dorothy) | m(ann, dorothy),mc(ann),pc(ann).
pc(dorothy) | f(brian, dorothy),mc(brian),pc(brian).
bt(ann)     | mc(ann), pc(ann).
bt(brian)   | mc(brian), pc(brian).
bt(dorothy) | mc(dorothy),pc(dorothy).
```

**Figure 3.4.** The grounded version of the *blood type* Bayesian logic program of Figure 3.3 where only clauses $c$ with $\text{head}(c) \in \text{LH}(B)$ and $\text{body}(c) \subset \text{LH}(B)$ are retained. It (directly) encodes the Bayesian network as shown in Figure 3.5. The structure of the Bayesian network coincides with the dependency graph of the *blood type* Bayesian logic program.

where all Bayesian predicates have *true* and *false* as states, and *noisy_or* as combining rule. The program specifies three conditional probability distributions for `fever` which are combined by the combining rule. In the dependency graph




*noisy or* $\{\mathbf{P}(\texttt{fever}|\texttt{flu}), \mathbf{P}(\texttt{fever}|\texttt{cold}), \mathbf{P}(\texttt{fever}|\texttt{malaria})\}$ is associated with `fever`, which is defined as on page 444 in [Russell and Norvig, 1995]. ○

Thus, if $DG(B)$ is acyclic and not empty, and every node in $DG(B)$ has a finite indegree then $DG(B)$ encodes a (possibly infinite) Bayesian network, because the least Herbrand model always exists and is unique. Consequently, the following independence assumption holds:

**Independence Assumption 3.13** (Dependency Graph) Each node `x` is independent of its non-descendants given a joint state of its parents $\mathbf{Pa}(\texttt{x})$ in the dependency graph. ○

For instance the dependency graph of the *blood type* program as shown in Figures 3.4 and 3.5 encodes that the random variable $\texttt{bt}(\texttt{dorothy})$ is independent from $\texttt{pc}(\texttt{ann})$ given a joint state of $\texttt{pc}(\texttt{dorothy}), \texttt{mc}(\texttt{dorothy})$. Using this assumption the following Theorem [taken from Kersting and De Raedt, 2001c] holds:

**Theorem 3.14** (Semantics) *Let $B$ be a Bayesian logic program. If*

(1) $\text{LH}(B) \neq \emptyset$,
(2) $\text{DG}(B)$ *is acyclic, and*
(3) *each node in* $\text{DG}(B)$ *is influenced by a finite set of random variables*

*then $B$ specifies a unique probability distribution $\mathbf{P}_B$ over* $\text{LH}(B)$. ○

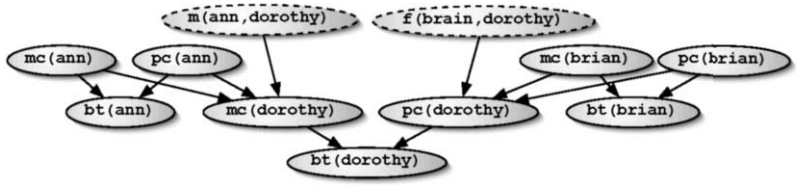



**Figure 3.5.** The structure of the Bayesian network represented by the grounded *blood type* Bayesian logic program in Figure 3.4. The structure of the Bayesian network coincides with the dependency graph. Omitting the dashed nodes yields the original Bayesian network of Figure 3.1.

*Proof sketch:* To see this, note that the least Herbrand LH($B$) always exists, is unique and countable. Thus, $DG(B)$ exists and is unique, and due to condition (3) the combined probability distribution for each node of $DG(B)$ is computable. Furthermore, because of condition (1) a total order $\pi$ on $DG(B)$ exists, so that one can see $B$ together with $\pi$ as a stochastic process over LH($B$). An induction argument over $\pi$ together with condition 2 allows one to conclude that the family of finite-dimensional distributions of the process is projective (cf. Bauer [1991]), i.e. , the joint probability distribution over each finite subset $\mathbf{S} \subseteq \mathrm{LH}(B)$ is uniquely defined and $\sum_y \mathbf{P}(\mathbf{S}, \mathtt{x} = y) = \mathbf{P}(\mathbf{S})$. Thus, the preconditions of *Kolmogorov's theorem* [Bauer, 1991, page 307] hold, and it follows that $B$ given $\pi$ specifies a probability distribution $\mathbf{P}$ over LH($B$). This proves the theorem because the total order $\pi$ used for the induction is arbitrary. □

A program $B$ satisfying the conditions (1)–(3) of Theorem 3.14 is called *well-defined*. A well-defined Bayesian logic program $B$ specifies a joint distribution over the random variables in the least Herbrand model LH($B$). As with Bayesian networks, the joint distribution over each self-contained [12], finite set $I \subseteq \mathrm{LH}(B)$ can be factored to

$$\mathbf{P}(I) = \prod_{\mathtt{x} \in I} \mathbf{P}(\mathtt{x} | \mathbf{Pa}(\mathtt{x}))$$

where the *parent* relation **Pa** is according to the dependency graph.

The *blood type* Bayesian logic program in Figure 3.3 is an example of a well-defined Bayesian logic program. Its grounded version is shown in Figure 3.4. It essentially encodes the original blood type Bayesian network of Figures 3.1 and 3.2. The only differences are the two predicates `m/2` and `f/2`, which can be in one of the logical states *true* and *false*. Using these predicates and an appropriate set of Bayesian facts (the 'extension') one can encode the Bayesian network for any family. This situation is akin to that in deductive databases, where the 'intension' (the clauses) encodes the overall regularities and the 'extension' (the facts) the specific context of interest. By interchanging the extension, one can swap contexts (in our case, families).

[12] A set $I$ is *self-contained* if $\forall \mathtt{x} \in I \ : \ \mathbf{Pa}(\mathtt{x}) \subseteq I$ where the *parent* relation **Pa** is according to the dependency graph.

### 3.2.3 Procedural Semantics

Clearly, any (conditional) probability distribution over random variables of the Bayesian network corresponding to the least Herbrand model can — in principle — be computed. As the least Herbrand model (and therefore the corresponding Bayesian network) can become (even infinitely) large, the question arises as to whether one needs to construct the full least Herbrand model (and Bayesian network) to be able to perform inferences. Here, inference means the process of answering probabilistic queries.

**Definition 3.15** (Probabilistic Query) A probabilistic query to a Bayesian logic program $B$ is an expression of the form

$$\texttt{?-}\ \mathtt{q}_1, \ldots, \mathtt{q}_n \mid \mathtt{e}_1 = e_1, \ldots, \mathtt{e}_m = e_m$$

where $n > 0$, $m \geq 0$. It asks for the conditional probability distribution

$$\mathbf{P}(\mathtt{q}_1, \ldots, \mathtt{q}_n \mid \mathtt{e}_1 = e_1, \ldots, \mathtt{e}_m = e_m)$$

of the query variables $\mathtt{q}_1, \ldots, \mathtt{q}_n$ where $\{\mathtt{q}_1, \ldots, \mathtt{q}_n, \mathtt{e}_1, \ldots, \mathtt{e}_m\} \subseteq \mathrm{HB}(B)$. ∘

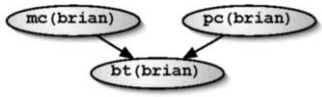
To answer a probabilistic query, one fortunately does not have to compute the complete least Herbrand model. It suffices to consider the so-called support network.

**Definition 3.16** (Support Network) The *support network* $N$ of a random variable $\mathtt{x} \in \mathrm{LH}(B)$ is defined as the induced subnetwork of

$$\{\mathtt{x}\} \cup \{\mathtt{y} \mid \mathtt{y} \in \mathrm{LH}(B) \text{ and } \mathtt{y} \text{ influences } \mathtt{x}\}\ .$$

The support network of a finite set $\{\mathtt{x}_1, \ldots, \mathtt{x}_k\} \subseteq \mathrm{LH}(B)$ is the union of the networks of each single $\mathtt{x}_i$. ∘

**Example 3.17** For instance, the support network for `bt(dorothy)` is the Bayesian network shown in Figure 3.5. The support network for `bt(brian)` is the subnetwork with root `bt(brian)`, i.e.




∘

That the support network of a finite set $\mathbf{X} \subseteq \mathrm{LH}(B)$ is sufficient to compute $\mathbf{P}(\mathbf{X})$ follows from the following Theorem [taken from Kersting and De Raedt, 2001c]:

**Theorem 3.18** (Support Network) *Let $N$ be a possibly infinite Bayesian network, let $\mathbf{Q}$ be nodes of $N$ and $\mathbf{E} = \boldsymbol{e}$, $\mathbf{E} \subset N$, be some evidence. The computation of $\mathbf{P}(\mathbf{Q} \mid \mathbf{E} = \boldsymbol{e})$ does not depend on any node $\mathbf{x}$ of $N$, which is not a member of the support network $N(\mathbf{Q} \cup \mathbf{E})$.* ∘

*Proof sketch:* In order to prove the theorem we only have to show that $N(\{Q_1,\ldots,Q_n\}\cup\mathbf{E})$ is sufficient to compute $\mathbf{p}(X_1,\ldots,X_l)$ for any set $\{X_1,\ldots,X_l\}$, $l>0$, of random variables in $N(\{Q_1,\ldots,Q_n\}\cup\mathbf{E})$. The theorem follows then from the definition of conditional probability density:

$$\mathbf{P}(Q_1,\ldots,Q_n \mid \mathbf{E}=\mathbf{e}) = \frac{\mathbf{P}(Q_1,\ldots,Q_n,\mathbf{E}=\mathbf{e})}{P(\mathbf{E}=\mathbf{e})} .$$

We proceed in a similar way to the proof of Theorem 3.14. Let $\pi$ be a total order of the nodes in $N$. Let $T$ be a (non-empty) index set, $\mathcal{H}(T)$ be the set of all non-empty, finite subsets of $T$, and $(A_n)_{n\in T}$ be the sequence of random variables in $N$ in ascending order to $\pi$. Analogously to the proof of Theorem 3.14 we can prove by induction that $N$ specifies a projective family of probability measure. Now, the set $\{Q_1,\ldots,Q_n\}\cup\mathbf{E}$ corresponds to $\mathbf{A}(\mathbf{H})$ for some $H\in\mathcal{H}(T)$, and the set $\{X_1,\ldots,X_l\}$ corresponds to $\mathbf{A}(\mathbf{L})$ for some $L\in\mathcal{H}(T)$. In order to compute $\mathbf{p}(X_1,\ldots,X_l)$ we consider the completion

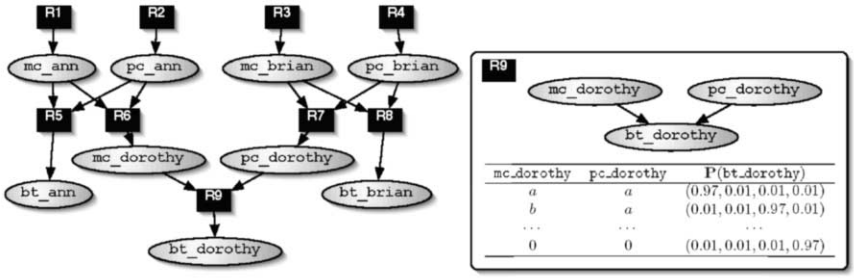
$$\mathbf{C}(\mathbf{L}) = \{D\in N \mid D \text{ influences some } X_i\in\mathbf{A}(\mathbf{L})\}$$

of $\mathbf{A}(\mathbf{L})$ (resp. $\mathbf{C}(\mathbf{H})$ of $\mathbf{A}(\mathbf{H})$). The set $\mathbf{C}(\mathbf{H})$ equals per definitionem the set of nodes of $N(\{Q_1,\ldots,Q_n\}\cup\mathbf{E})$. Therefore, we have $\mathbf{A}(\mathbf{L})\subset\mathbf{C}(\mathbf{H})$ and, hence, $\mathbf{C}(\mathbf{L})\subset\mathbf{C}(\mathbf{H})$. As in the proof of Theorem 3.14, the probability densities over $\mathbf{C}(\mathbf{H})$ (resp. $\mathbf{C}(\mathbf{L})$) are specified by a unique Bayesian network $N(H)$ (resp. $N(L)$). It consists of all random variables in $\mathbf{C}(\mathbf{H})$ (resp. $\mathbf{C}(\mathbf{L})$) and of all edges between nodes in $N$, which are random variables in $\mathbf{C}(\mathbf{H})$ (resp. $\mathbf{C}(\mathbf{L})$). Since $N$ specifies a projective family of probability measures, $N(L)$ is a subnetwork of $N(H)$. That means the computation of $\mathbf{p}(X_1,\ldots,X_l)$ only depends on nodes and edges in $N(H)$. But $N(H)$ is per definitionem the support network $N(\{Q_1,\ldots,Q_n\}\cup\mathbf{E})$. This proves the theorem. □

To compute the support network $N(\{\mathtt{q}\})$ of a single variable `q` efficiently, let us look at logic programs from a proof theoretic perspective. From this perspective, a logic program can be used to prove that certain atoms or goals (see end of Section 2.1) are logically entailed by the program. The set of all proofs of `:-bt(dorothy)` captures all information needed to compute $N(\{\mathtt{bt(dorothy)}\})$. More exactly, the set of all ground clauses employed to prove `bt(dorothy)` constitutes the families of the support network $N(\{\mathtt{bt(dorothy)}\})$. For `:-bt(dorothy)`, they are the ground clauses shown in Figure 3.4. To build the support network, we only have to gather all ground clauses used to prove the query variable and have to combine multiple copies of ground clauses with the same head using corresponding combining rules. To summarize, the support network $N(\{\mathtt{q}\})$ can be computed as follows:

1: Compute all proofs for `:-q`.
2: Extract the set $S$ of ground clauses used to prove `:-q`.
3: Combine multiple copies of ground clauses $\mathtt{h}|\mathtt{b}\in S$ with the same head `h` using combining rules.

**Example 3.19** Applying this to `:-bt(dorothy)` yields the support network as shown in Figure 3.5. ○




_dorothy|mc_dorothy, pc_dorothy with associated conditional probability table.

The method can easily be extended to compute the support network for $\mathbf{P}(\mathbf{Q} \mid \mathbf{E} = \boldsymbol{e})$ . We simply compute all proofs of :-q, $\mathtt{q} \in \mathbf{Q}$, and :-e, $\mathtt{e} \in \mathbf{E}$ . The resulting support network can be fed into any (exact or approximative) Bayesian network engine to compute the resulting (conditional) probability distribution of the query. To minimize the size of the support network, one might also apply Schachter's Bayes' Ball algorithm [1998].

## 3.3 Extensions of the Basic Framework

So far, we described the basic Bayesian logic programming framework and defined the semantics of Bayesian logic programs. Various useful extensions and modifications are possible. In this section, we will introduce a *graphical representation* and will discuss *aggregate functions* and the efficient treatment of *logical atoms*. At the same time, we will also present further examples of Bayesian logic programs such as hidden Markov models [Rabiner, 1989] and probabilistic grammars [Manning and Schütze, 1999].

### 3.3.1 Graphical Representation

Bayesian logic programs have so far been introduced using an adaption of a logic programming syntax. Bayesian networks are, however, also graphical models and owe at least part of their popularity to their intuitively appealing graphical notation [Jordan, 1999]. Inspired on Bayesian networks, we develop in this section a graphical notation for Bayesian logic programs.

In order to develop a graphical representation for Bayesian logic programs, let us first consider a more redundant representation for Bayesian networks: augmented bipartite (directed acyclic) graphs as shown in Figure 3.6. In a bipartite graph, the set of nodes is composed of two disjoint sets such that no two nodes within the same set are adjacent. There are two types of nodes, namely (1) *gradient gray ovals* denoting random variables, and (2) *black boxes* denoting local probability models. There is a box for each family $\mathbf{Fa}(\mathtt{x_i})$ in the Bayesian network. The incoming edges refer to

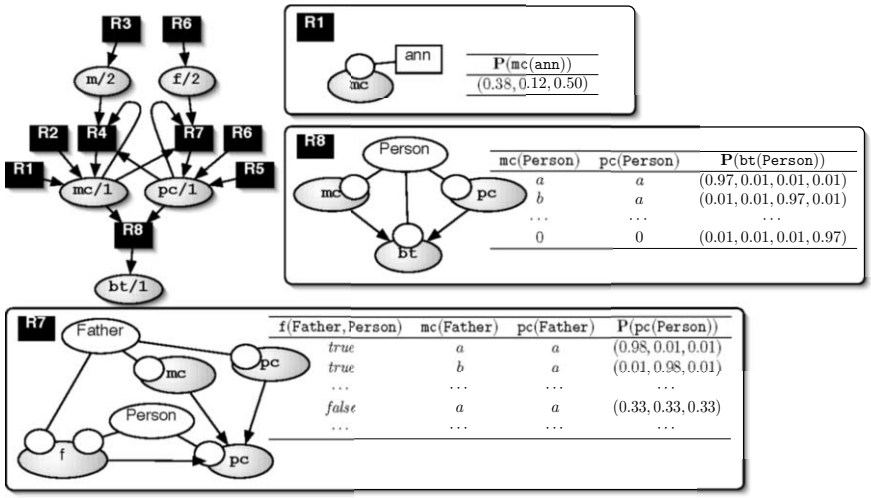



**Figure 3.7.** Graphical representation of the *blood type* Bayesian logic program. On the right-hand side, some local probability models associated with Bayesian clause nodes are shown, e.g., the Bayesian clause $R7$ $\mathtt{pc(Person)}|\mathtt{f(Father, Person)}, \mathtt{mc(Father)}, \mathtt{pc(Father)}$ with associated conditional probability distribution. For the sake of simplicity, not all Bayesian clauses are shown.

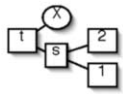
the parents $\mathbf{Pa}(\mathtt{x_i})$; the single outgoing edge points to $X_i$. Each box is augmented with a Bayesian network fragment specifying the conditional probability distribution $\mathbf{P}(\mathtt{x_i}|\mathbf{Pa}(\mathtt{x_i}))$.

**Example 3.20** In Figure 3.6, the fragment associated with $R9$ specifies the conditional probability distribution of $\mathbf{P}(\mathtt{bt(dorothy)}|\mathtt{mc(dorothy)}, \mathtt{pc(dorothy)})$. ○

Interpreting this as a propositional Bayesian logic program, the graph can be viewed as a *rule graph* known from database theory. Ovals represent Bayesian predicates, and boxes denote Bayesian clauses. More precisely, given a (propositional) Bayesian logic program $B$ with Bayesian clauses $R_i \equiv \mathtt{h_i}|\mathtt{b_{i_1}}, \ldots, \mathtt{b_{i_n}}$ , there are edges from from $R_i$ to $\mathtt{h_i}$ and from $\mathtt{b_{i_j}}$ to $R_i$ . Furthermore, to each Bayesian clause node, we associate the corresponding Bayesian clause as a Bayesian network fragment. Indeed, the graphical model in Figure 3.6 represents the propositional Bayesian logic program of Figure 3.4.

In order to represent first order Bayesian logic programs graphically, we have to encode Bayesian atoms and their variable bindings in the associated local probability models. Indeed, logical terms can naturally be represented as trees.

**Example 3.21** The term $\mathtt{t(s(1,2), X)}$ corresponds to the tree




○

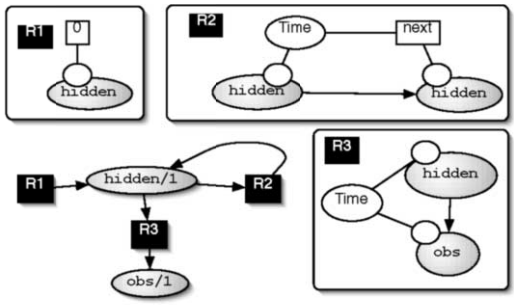
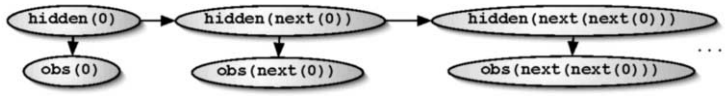



**Figure 3.8.** A Bayesian logic program modeling a hidden Markov model. The functor `next/1` is used to encode the discrete time. The clausal representation is given in the text.

Logical variables such as `X` are encoded as white ovals. Constants and functors such as `1`, `2`, `s`, and `t` are represented as white boxes. Bayesian atoms are represented as gradient gray ovals containing the predicate name such as `pc`. Arguments of atoms are treated as placeholders for terms. They are represented as white circles on the boundary of the ovals (ordered from left to right). The term appearing in the argument is represented by an undirected edge between the white oval representing the argument and the 'root' of the tree encoding the term (we start in the argument and follow the tree until reaching variables).

**Example 3.22** Consider the Bayesian logic program in Figure 3.7. It models the *blood type* domain. The graphical representation conveys the meaning of the Bayesian clause $R7$: *the paternal genetic information* `pc(P)` *of a person* `P` *is influenced by the maternal* `mc(F)` *and the paternal* `pc(F)` *genetic information of the* `P`*'s father* `F`. ○

As another example, consider Figure 3.8, which illustrates the use of functors to represent dynamic probabilistic models. More precisely, it shows a hidden Markov model (HMM) [Rabiner, 1989]. HMMs are extremely popular for analyzing sequential data. Application areas include computational biology, user modeling, speech recognition, empirical natural language processing, and robotics. At each `Time`, the system is in a state `hidden(Time)`. The apriori probability of being in some state is quantified by the Bayesian fact $R1$ `hidden(0)` . The time-independent probability of being in some state at time `next(Time)` given that the system was in a state at `TimePoint` is captured in the Bayesian clause $R2$ `hidden(next(Time))` | `hidden(Time)` . Here, the next time point is represented as functor `next/1` . In HMMs, however, we do not have direct access to the states `hidden(Time)`. Instead, we measure some properties `obs(Time)` of the states. The measurement is quantified in Bayesian clause $R3$ obs(`Time`) | `hidden(time)` . The dependency graph of the Bayesian logic program directly encodes the well-known Bayesian network structure of HMMs:



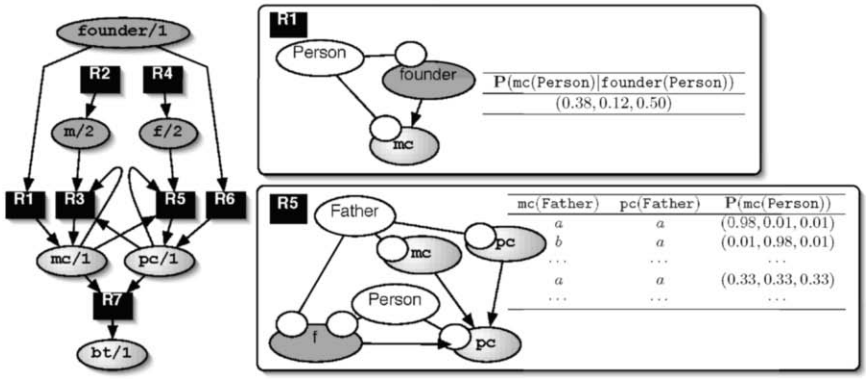



**Figure 3.9.** The *blood type* Bayesian logic program distinguishing between *Bayesian* (gradient gray ovals) and *logical* atoms (solid gray ovals).

### 3.3.2 Logical Atoms

Reconsider the *blood type* Bayesian logic program in Figure 3.7. The `mother/2` and `father/2` relations are not really *random* variables but *logical* ones because they are always in the same state, namely *true*, with probability 1 and can depend only on other logical atoms. These predicates form a kind of logical background theory. Therefore, when predicates are declared to be *logical*, one need not represent them in the conditional probability distributions. Consider the *blood type* Bayesian logic program in Figure 3.9. Here, `mother/2` and `father/2` are declared to be *logical*. Consequently, the conditional probability distribution associated with the definition of, e.g., `pc/1` takes only `pc(Father)` and `mc(Father)` into account but not `f(Father,Person)`. It is applied only to those substitutions for which `f(Father,Person)` is true, i.e., in the least Herbrand model. This can efficiently be checked using any Prolog engine. Furthermore, one may omit these logical atoms from the induced support network. More importantly, logical predicates provide the user with the full power of Prolog. In the *blood type* Bayesian logic program of Figure 3.9, the logical background knowledge defines the `founder/1` relation as

```
founder(Person):-\+(mother(_,Person);father(_,Person)).
```

Here, `\+` denotes *negation*, the symbol `_` represents an anonymous variable, which is treated as new, distinct variable each time it is encountered, and the semicolon denotes a *disjunction*. The rest of the Bayesian logic program is essentially as in Figure 3.3. Instead of explicitly listing `pc(ann),mc(ann),pc(brian),mc(brian)` in the extensional part we have `pc(P)|founder(P)` and `mc(P)|founder(P)` in the intensional part.

The full power of Prolog is also useful to elegantly encode dynamic probabilistic models. Figure 3.10 **(a)** shows the generic structure of an HMM where the discrete time is now encoded as `next/2` in the logical background theory using standard Prolog

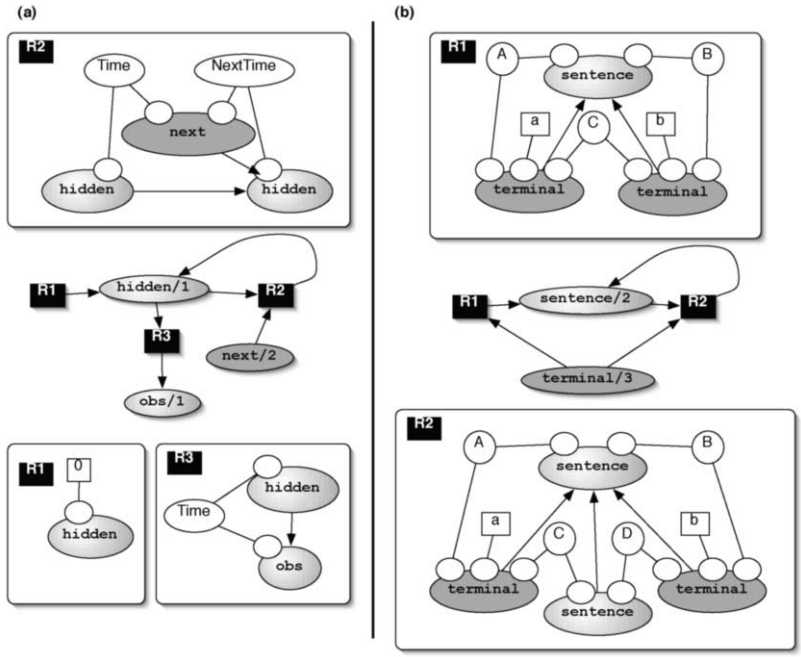



**Figure 3.10.** Two dynamic Bayesian logic programs. **(a)** The generic structure of a hidden Markov model more elegantly represented as in Figure 3.8 using $\texttt{next}(\texttt{X},\texttt{Y}) : -\texttt{integer}(\texttt{Y}), \texttt{Y} > 0, \texttt{X is Y} - 1.$. **(b)** A probabilistic context-free grammar over $\{a^n b^n\}$. The logical background theory defines `terminal/3` as $\texttt{terminal}([\texttt{A}|\texttt{B}], \texttt{A}, \texttt{B})$.

predicates:

$$\texttt{next}(\texttt{X},\texttt{Y}) \texttt{:-} \texttt{integer}(\texttt{Y}), \texttt{Y} > 0, \texttt{X is Y} - 1.$$

Prolog's predefined predicates (such as `integer/1`) avoid a cumbersome representation of the dynamics via the successor functor $0, \texttt{next}(0), \texttt{next}(\texttt{next}(0)), \ldots$ Imagine querying `?- obs(100)` using the successor functor,

$$\texttt{?- obs}(\texttt{next}(\texttt{next}(\ldots(\texttt{next}(0))\ldots))) \,.$$

Whereas HMMs define probability distributions over regular languages, *probabilistic context-free grammars* (PCFGs) [Manning and Schütze, 1999] define probability distributions over context-free languages. Application areas of PCFGs include e.g. natural language processing and computational biology. For instance, mRNA sequences constitute context-free languages.

**Example 3.23** Consider e.g. the following PCFG

1.0 : terminal([A|B], A, B).

0.3 : sentence(A, B):-terminal(A, a, C), terminal(C, b, B).

0.7 : sentence(A, B):-terminal(A, a, C), sentence(C, D), terminal(D, b, B).

defining a distribution over $\{a^n b^n\}$ . The grammar is represented as probabilistic *definite clause grammar* where the terminal symbols are encoded in the logical background theory via the first rule terminal([A|B], A, B) . ○

Recall from Section 2.3.2 that a PCFG defines a stochastic process with leftmost rewriting, i.e., refutation steps as transitions. Words, say *aabb*, are parsed by querying ?- sentence([a, a, b, b], []). The third rule yields ?- terminal([a, a, b, b], a, C), sentence(C, D), terminal(D, b, []). Applying the first rule yields ?- sentence([a, b, b], D), terminal(D, b, []) and the second rule ?- terminal([a, b, b], a, C), terminal(C, b, D), terminal(D, b, []). Applying the first rule three times yields a successful refutation. The probability of a refutation is the product of the probability values associated with clauses used in the refutation; in our case $0.7 \cdot 0.3$. The probability of *aabb* then is the sum of the probabilities of all successful refutations. This is also the basic idea underlying Muggleton's *stochastic logic programs* [1996], which we also encountered in Section 2.3.2.

Figure 3.10 **(b)** shows the $\{a^n b^n\}$ PCFG of Example 3.23 represented as a Bayesian logic program. Its underlying logic program coincides with the *definite clause grammar* shown in Example 3.23. In contrast to the PCFGs representation, which associate a single probability value only with each clause, we associate a complete conditional probability distribution

| **P**(sentence(A, B)\| *terminal*(*A*, *a*, *C*), *terminal*(*C*, *b*, *B*)) |
|---|
| (0.3, 0.7) |

| **P**(sentence(A, B)\| *terminal*(*A*, *a*, *C*), sentence(C, D), *terminal*(*D*, *b*, *B*)) |
|---|
| (0.7, 0.3) |
| (0.0, 1.0) |

where logical atoms are written in italic. For the query ?- sentence([a, a, b, b], []), the following Bayesian network


is induced where we omitted logical nodes.

### 3.3.3 Aggregate Functions

An alternative to combining rules are *aggregate functions*. As an example, consider the *university* domain due to Getoor et al. [2001].

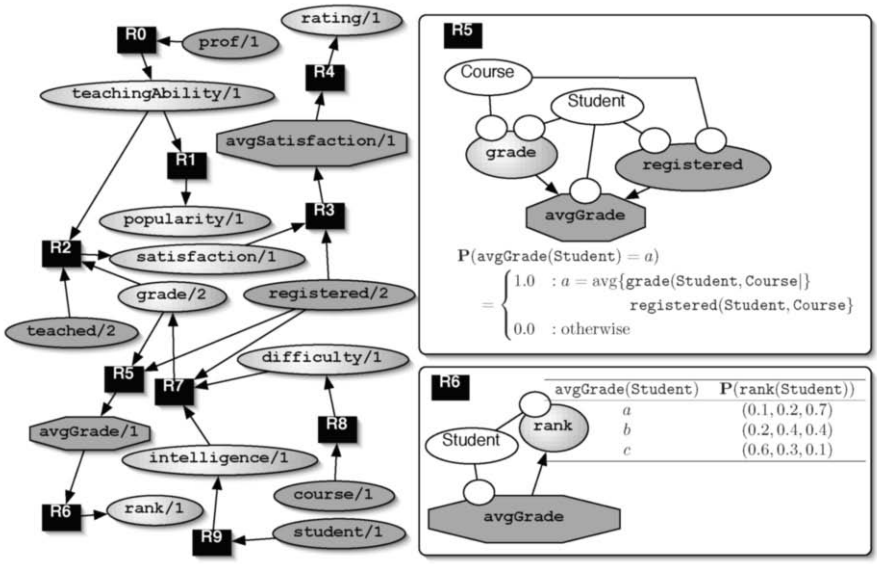



**Figure 3.11.** The Bayesian logic program for the *university* domain. Octagonal nodes denote aggregate predicates and atoms.

**Example 3.24** The university domain contains professors, students, courses, and course registrations. Objects in this domain have several descriptive attributes such as `intelligence/1` and `rank/1` of a `student/1`. A student will typically be registered in several courses; the student's rank depends on the grades she receives in all of them. So, we have to specify a probabilistic dependence of the student's rank on a multiset of course grades of size 1, 2, and so on. ○

In this situation, the notion of aggregation is more appropriate than that of a combining rule. Using combining rules, the Bayesian clauses would describe the dependence for a single course only. All information of how the rank probabilistically depends on the multiset of course grades would be 'hidden' in the combining rule. In contrast, when using an aggregate function, the dependence is interpreted as a probabilistic dependence of `rank` on some *deterministically* computed aggregate property of the multiset of course grades. The probabilistic dependence is moved out of the combining rule.

To model this, we introduce *aggregate* predicates. They represent deterministic random variables, i.e., the state of an aggregate atom is a function of the joint state of its parents.

**Example 3.25** The Bayesian logic program shown in Figure 3.11 encodes the *university* domain. Here, `avgGrade/1` is an aggregate predicate, denoted as a octagonal node. As combining rule, the average of the parents' states is deterministically computed, cf. Bayesian clause $R5$. In turn, the student's `rank/1` probabilistically depends on her averaged rank, cf. $R6$. ○

The use of aggregate functions is inspired by *probabilistic relational models* [Pfeffer, 2000]. As we will show in the related work section, using aggregates in Bayesian logic programs, it is easy to model probabilistic relational models.

§ 4

# Learning Bayesian Logic Programs *

*... in which the problem of learning Bayesian logic programs from data is defined, solution techniques for structure learning and for parameter estimation are presented and experimentally evaluated ...*

So far, we have assumed that there is an expert who provides both the structure and the conditional probability distributions of the Bayesian logic program. This is not always easy. Extracting knowledge from a human expert and representing that knowledge has been proven to be an difficult and labor intensive effort [Forsyth, 1994] and is also known as the *knowledge acquisition bottleneck*. Often, there is even no-one possessing the necessary expertise or knowledge. Instead of an expert, however, we may have access to data and a machine learning algorithm that automatically induces Bayesian clauses from the given data.

## 4.1 The Learning Setting: Probabilistic Learning from Interpretations

Learning Bayesian logic programs from data falls into the probabilistic learning from interpretations setting, which we have introduced in Section 2.3.1. To see why this is the case, let us analyze the types of data generated by Bayesian logic programs.

Let $B$ be a Bayesian logic program $B$ consisting of a set of intensional clauses $I$ and extensional facts $E$. According to the semantics, the Bayesian logic program specifies a probability distribution over the possible states of the random variables in the LH($B$). Thus, a data case generated by a Bayesian logic program consists of two components: a *logical* one, which is a Herbrand interpretation of the underlying logic program, and a *probabilistic* one, which is an assignment of states to the random variables of the interpretation. By analogy with the traditional Bayesian network learning setting [Heckerman, 1995], we do not require that the assignment of states to random variables is complete, i.e. certain states could be unobserved.

**Example 4.1** Example data cases are

$$D_1 = \{\mathtt{m}(\mathtt{cecily},\mathtt{fred}) = \mathit{true}, \mathtt{f}(\mathtt{henry},\mathtt{fred}) = \mathit{true}, \mathtt{pc}(\mathtt{cecily}) = a,$$
$$\mathtt{pc}(\mathtt{henry}) = b, \mathtt{pc}(\mathtt{fred}) = ?, \mathtt{mc}(\mathtt{cecily}) = b, \mathtt{mc}(\mathtt{henry}) = b,$$
$$\mathtt{mc}(\mathtt{fred}) = ?, \mathtt{bt}(\mathtt{cecily}) = ab, \mathtt{bt}(\mathtt{henry}) = b, \mathtt{bt}(\mathtt{fred}) = ?\},$$

* Builds on [Kersting and De Raedt, 2001a,c, 2002, Fischer and Kersting, 2003].

$$D_2 = \{\texttt{m(ann,dorothy)} = \mathit{true}, \texttt{f(brian,dorothy)} = \mathit{true}, \texttt{pc(ann)} = b,$$
$$\texttt{mc(ann)} = ?, \texttt{mc(brian)} = a, \texttt{mc(dorothy)} = a, \texttt{pc(dorothy)} = a,$$
$$\texttt{pc(brian)} = ?, \texttt{bt(ann)} = ab, \texttt{bt(brian)} = ?, \texttt{bt(dorothy)} = a\},$$

where '?' stands for an unobserved state. ○

Notice that — for ease of writing — we merged the two components of a data case into one. Indeed, the *logical part* $\mathrm{RandVar}(D_i)$ of a data case $D_i$ is a Herbrand interpretation, e.g.,

$$\mathrm{RandVar}(D_1) = \{\texttt{m(cecily,fred)}, \texttt{f(henry,fred)}, \texttt{pc(cecily)},$$
$$\texttt{pc(henry)}, \texttt{pc(fred)}, \texttt{mc(cecily)}, \texttt{mc(henry)},$$
$$\texttt{mc(fred)}, \texttt{bt(cecily)}, \texttt{bt(henry)}, \texttt{bt(fred)}\},$$

$$\mathrm{RandVar}(D_2) = \{\texttt{m(ann,dorothy)}, \texttt{f(brian,dorothy)}, \texttt{pc(ann)},$$
$$\texttt{mc(ann)}, \texttt{mc(brian)}, \texttt{mc(dorothy)}, \texttt{pc(dorothy)},$$
$$\texttt{pc(brian)}, \texttt{bt(ann)}, \texttt{bt(brian)}, \texttt{bt(dorothy)}\},$$

At this point, it is important to realize that these two Herbrand interpretations correspond to two different extensions.

**Example 4.2** Indeed, it holds that $\mathrm{RandVar}(D_1) = \mathrm{LH}(I \cup E_1)$ and $\mathrm{RandVar}(D_2) = \mathrm{LH}(I \cup E_2)$, where $I$ could be

```
mc(X)  | m(Y,X),mc(Y),pc(Y).
pc(X)  | f(Y,X),mc(Y),pc(Y).
bt(X)  | mc(X),pc(X).
```

and the $E_i$ could be :

$$E_1 = \{\texttt{m(cecily,fred)}, \texttt{f(henry,fred)}, \texttt{pc(cecily)}, \texttt{pc(henry)},$$
$$\texttt{mc(cecily)}, \texttt{mc(henry)}\},$$
$$E_2 = \{\texttt{m(ann,dorothy)}, \texttt{f(brian,dorothy)}, \texttt{pc(ann)}, \texttt{pc(brian)}$$
$$\texttt{mc(ann)}, \texttt{mc(brian)}\},$$

○

It is required that each of the given Herbrand interpretations $D_i$ is a model for the set of intensional clauses $I$. This condition has been called *logical validity*. It holds whenever $\mathrm{LH}(D \cup I) = D$. At this point, the reader may want to verify that this condition holds for the two Herbrand interpretations $\mathrm{RandVar}(D_i)$ specified above and $I$. The reader should also observe that the logical part of a data case is a *complete* model of the target Bayesian logic program such as the data cases $D_1$ and $D_2$ (see above) and not a *partial* one[13]. This is motivated by 1) the analogy with Bayesian

[13] Partial models specify the truth-value of *some* of the elements in the Herbrand base. Working with partial models in the logical sense would amount to having incomplete knowledge of the set of relevant random variables in data cases. This is akin to the serious problem of detecting hidden variables in the context of Bayesian network learning [Kwoh and Gillies, 1996, Elidan and Friedman, 2001] and to multiple predicate learning [De Raedt et al., 1993] and even predicate invention [Stahl, 1993] in ILP.

network learning and 2) the problems with learning from partial models in ILP. First, data cases as they have been used in Bayesian network learning are the propositional equivalent of the data cases that we introduced above. Second, it is well-known that learning from partial models is harder than learning from complete models, see e.g. [De Raedt, 1997]. These two points also clarify why the semantics of the set of relevant random variables coincides with the least Herbrand model and at the same time why we do not restrict the set of states of Bayesian predicates to $\{true, false\}$.

We are now able to formally define the notion of a data case for Bayesian logic programs:

**Definition 4.3** (Data Case) A data case $D$ for an intensional Bayesian logic program $B$ consists of a **logical part**, which is a Herbrand interpretation $\mathrm{RandVar}(D)$ such that $\mathrm{RandVar}(D) = \mathrm{LH}(B \cup \mathrm{RandVar}(D))$, and a **probabilistic part**, which is a partially observed joint state of $\mathrm{RandVar}(D)$, i.e., an assignment of states to some of the random variables in $\mathrm{RandVar}(D)$. ∘

There is one further logical constraint to take into account while learning Bayesian logic programs. It is concerned with the acyclicity requirement (cf. property 2 in Theorem 3.14) imposed on Bayesian logic programs. It is therefore required that for each data case $D$ the induced Bayesian network over $\mathrm{LH}(B \cup \mathrm{RandVar}(D))$ is acyclic.

By now, we can instantiate the probabilistic ILP learning setting in Definition 2.26 for learning Bayesian logic programs.

**Definition 4.4** (Learning Problem) **Given** a set $\mathbf{D} = \{D_1, \ldots, D_m\}$ of data cases, which are sampled i.i.d., a set $\mathcal{H}$ of sets of Bayesian clauses (according to some syntactic language bias), a scoring function $\mathrm{score}_{\mathbf{D}} : \mathcal{H} \mapsto \mathbb{R}$, and a combining rule for each Bayesian predicate, **find** a hypothesis $H^* \in \mathcal{H}$ such that for all $D_i \in \mathbf{D} : \mathrm{LH}(H^* \cup \mathrm{RandVar}(D_i)) = \mathrm{RandVar}(D_i)$, $H^*$ is acyclic on the data, and $H^*$ maximizes $score_{\mathbf{D}}$. ∘

Thus, we investigate the learning of Bayesian logic programs from possible interpretations only.

The *hypothesis space* $\mathcal{H}$ to be explored consists of intensional Bayesian logic programs, i.e. finite sets of Bayesian clauses to which conditional probability distributions are associated. As score, we will use the likelihood, which will be defined in Section 4.3, but other scores such as *minimum description length* variants are also possible. Finally, the syntactic bias $\mathcal{L}_1$ that we will employ in the experimental section is as follows:

**Specifiction 4.5** (Syntactic Bias $\mathcal{L}_1$) The clauses contained in $\mathcal{H}$ are range-restricted, constant-free and contain only predicate symbols that occur in one of the data cases. The bodies consist of at most three atoms. Furthermore, each predicate is defined by a single clause. ∘

**Example 4.6** For the syntactic bias $\mathcal{L}_1$, legal clauses for the Bayesian predicate `mc/1` include `mc(X)|m(X,X)`; `mc(X)|m(Y,X)`; `mc(X)|m(X,Y)`; `mc(X)|mc(X)`; ...; `mc(X)|pc(X)`; ...; `mc(X)|m(X,X),mc(X)`; `mc(X)|m(Y,X),mc(Y)`; `mc(X)|m(X,Y),mc(Y)`; and so on. ∘

Following the general approach for solving the probabilistic ILP learning problem described in Section 2.4, the problem of learning Bayesian networks can be divided into two subproblems: parameter estimation and structure learning. The next two sections deal with structure learning and parameter estimation, respectively, for Bayesian logic programs.

## 4.2 Scooby – Structural learning of intensional Bayesian logic programs

Scooby (**s**tru**c**tural learning **o**f intensi**o**nal **B**a**y**esian logic programs), cf. Algorithm II.1, is an algorithm for solving the learning problem. Roughly speaking, Scooby performs a heuristic search using traditional ILP refinement operators on clauses, cf. Definition 2.13. The hypothesis currently under consideration is evaluated using some score as heuristic. The hypothesis that scores best is selected as the final hypothesis.

We will illustrate how Scooby works for the special case of Bayesian networks. It turns out that for the special case of Bayesian networks, Scooby coincides with well-known and effective score-based techniques for learning Bayesian networks [Heckerman, 1995]. In a next step, we will show that Scooby works for first order Bayesian logic programs, too. For the sake of readability, we will — in this section — assume that there exists a method to estimate the parameters of a Bayesian logic program with a fixed structure using the scoring function. Algorithms to realize this for the likelihood function will be introduced in Section 4.3. Finally, we will discuss the basic principles only. Extensions are possible and will be discussed in Section 4.2.3.

Let us first explain who Scooby works on Bayesian networks.

### 4.2.1 The Propositional Case: Bayesian Networks

Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a fixed set of random variables. The set $\mathbf{X}$ corresponds to the least Herbrand model $\{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$ of an unknown propositional Bayesian logic program representing a Bayesian network. The probabilistic dependencies among the relevant random variables are unknown, i.e. the propositional Bayesian clauses are unknown. Therefore, we will employ such a propositional Bayesian logic program as a *candidate* and estimate its parameters. Assume the data cases $\mathbf{D} = \{D_1, \ldots, D_m\}$ look like

$$\{\mathtt{m}(\mathtt{ann}, \mathtt{dorothy}) = \mathit{true}, \mathtt{f}(\mathtt{brian}, \mathtt{dorothy}) = \mathit{true}, \mathtt{pc}(\mathtt{ann}) = a,$$
$$\mathtt{mc}(\mathtt{ann}) = ?, \mathtt{mc}(\mathtt{brian}) = ?, \mathtt{mc}(\mathtt{dorothy}) = a, \mathtt{mc}(\mathtt{dorothy}) = a,$$
$$\mathtt{pc}(\mathtt{brian}) = b, \mathtt{bt}(\mathtt{ann}) = a, \mathtt{bt}(\mathtt{brian}) = ?, \mathtt{bt}(\mathtt{dorothy}) = a\},$$

which is a data case for the Bayesian network in Figure 3.1. (Note, that — in this example — the atoms have to be interpreted as propositions). Each $H$ in the hypothesis space $\mathcal{H}$ is a Bayesian logic program consisting of $n$ propositional clauses: for each $X_i \in \mathbf{X}$ a single clause $c$ with $\mathit{head}(c) = \mathtt{x_i}$ and $\mathit{body}(c) \subseteq \{\mathtt{x_1}, \ldots, \mathtt{x_n}\} \setminus \{\mathtt{x_i}\}$. To traverse $\mathcal{H}$ we specify two *refinement* operators $\rho_g : \mathcal{H} \mapsto 2^{\mathcal{H}}$ and $\rho_s : \mathcal{H} \mapsto 2^{\mathcal{H}}$,

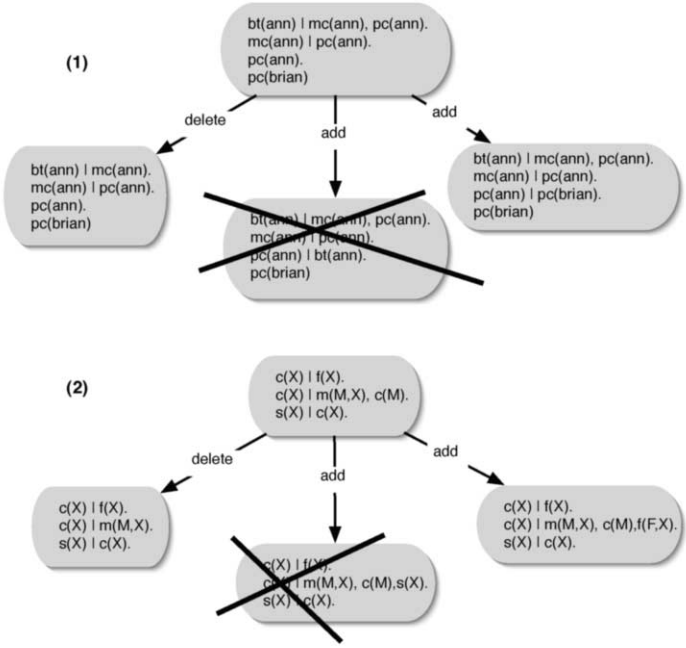



**Figure 4.1.** (1) The use of refinement operators during structural search for Bayesian networks. We can add ($\rho_s$) a proposition to the body of a clause or delete ($\rho_g$) it from the body. (2) The use of refinement operators during structural search within the framework of Bayesian logic programs. We can add ($\rho_s$) an atom to the body of a clause or delete ($\rho_g$) it from the body. Candidates crossed out in (1) and (2) are illegal because they are cyclic. Other refinement operators are reasonable such as adding or deleting logically valid clauses.

that take a hypothesis and modify it to produce a set of possible candidates. In the case of Bayesian networks, the operator $\rho_g(H)$ deletes a Bayesian proposition from the body of a Bayesian clause $c_i \in H$, and the operator $\rho_s(H)$ adds a Bayesian proposition to the body of $c_i$ (cf. Figure 4.1). The search algorithm performs a greedy, informed search in $\mathcal{H}$ based on $score_{\mathbf{D}}$.

As a simple illustration of SCOOBY for learning Bayesian networks we consider a greedy hill-climbing algorithm incorporating $score_{\mathbf{D}}(H) := LL(\mathbf{D}, H)$, the log-likelihood of the data $\mathbf{D}$ given a candidate structure $H$. This means that we compute for each fixed candidate structure the maximum likelihood parameters. We pick an initial candidate $H \in \mathcal{H}$ as starting point and score it. This means, we compute those parameters of the conditional probability distributions associated with clauses in $H$ that maximize the likelihood $LL(\mathbf{D}, H)$. The fully unconnected Bayesian network, i.e., the set of propositions, would be an obvious starting point. Then, we use $\rho_g(H)$ and $\rho_s(H)$ to compute the legal "neighbours" (candidates being acyclic) of $H$ in $\mathcal{H}$ and

**Algorithm II.1**: A simplified skeleton of a greedy algorithm SCOOBY($H, \mathbf{D}$) for **s**tru**c**tural learning **o**f intensi**o**nal **Ba**y**e**sian logic programs). Note that we have omitted the initialization of the conditional probability distributions associated with Bayesian clauses with random values. The operators $\rho_g$ and $\rho_s$ are generalization and specialization operators.

**input** : $H$, a (valid) Bayesian logic program; $\mathbf{D}$, a finite set of data cases
**output**: a modified Bayesian logic program

```
repeat
    H' := H
    foreach H'' ∈ ρ_g(H') ∪ ρ_s(H') do
        if H'' is (logically) valid on D then
            if the Bayesian network induced by H'' on the data is acyclic then
                if score_D(H'') > score_D(H) then
                    H := H''
until score_D(H') = score_D(H)
return H
```

score them. All neighbours are valid (see below for a definition of validity). For instance, replacing `pc(ann)` with `pc(ann)|pc(brian)` gives such a "neighbour". We take that $H' \in \rho_g(H) \cup \rho_s(H)$ that yields the best improvement in score. This process is continued until no further improvement in score is obtained.

### 4.2.2 The First Order Case: Bayesian Logic Programs

Let us now explain how SCOOBY works in the first order case. The key differences with the propositional case are (1) that some Bayesian logic programs will be logically invalid, and (2) that the traditional first order refinement operators must be used. Let us discuss both differences in turn.

**Logically Valid Hypotheses:** Difference (1) is the most important one, because it determines the legal candidate Bayesian logic programs. To account for this difference, two modifications to the traditional Bayesian network algorithm are needed.

The first modification concerns the initialization phase. There we must start from a logically valid and acyclic Bayesian logic program. Such a program can be computed using a CLAUDIEN like procedure [De Raedt and Bruynooghe, 1993, De Raedt and Dehaspe, 1997]. CLAUDIEN is an ILP-engine that computes a set $\mathcal{C}$ of logically valid clauses from a set of data cases. Furthermore, all clauses in $\mathcal{C}$ will be maximally general (w.r.t. $\theta$-subsumption), and CLAUDIEN will compute all such clauses (within the language bias $\mathcal{L}$). This implies that none of the clauses in $\mathcal{C}$ can be generalized without violating the logical validity requirement or the language bias $\mathcal{L}$.

**Example 4.7** Consider again the data cases

$$\begin{aligned}
D_1 =\{&\mathtt{m(cecily,fred)} = \mathit{true}, \mathtt{f(henry,fred)} = \mathit{true}, \mathtt{pc(cecily)} = a,\\
&\mathtt{pc(henry)} = b, \mathtt{pc(fred)} =?, \mathtt{mc(cecily)} = b, \mathtt{mc(henry)} = b,\\
&\mathtt{mc(fred)} =?, \mathtt{bt(cecily)} = ab, \mathtt{bt(henry)} = b, \mathtt{bt(fred)} =?\},\\
D_2 =\{&\mathtt{m(ann,dorothy)} = \mathit{true}, \mathtt{f(brian,dorothy)} = \mathit{true}, \mathtt{pc(ann)} = b,\\
&\mathtt{mc(ann)} =?, \mathtt{mc(brian)} = a, \mathtt{mc(dorothy)} = a, \mathtt{pc(dorothy)} = a,\\
&\mathtt{pc(brian)} =?, \mathtt{bt(ann)} = ab, \mathtt{bt(brian)} =?, \mathtt{bt(dorothy)} = a\},
\end{aligned}$$

The fact $\mathtt{bt(X)}$ is not allowed according to our language bias (it is not range-restricted). The clause $\mathtt{bt(X)|mc(X), pc(X)}$ is valid but not maximally general because the atom $\mathtt{pc(X)}$ can be deleted without violating the logical validity requirement. Any hypothesis including $\mathtt{m(X,Y)|mc(X), pc(Y)}$ would be logically invalid because Cecily is not the mother of Henry. Examples of maximally general clauses are $\mathtt{mc(X)|m(Y,X)}$, $\mathtt{pc(X)|f(Y,X)}$, $\mathtt{bt(X)|mc(X)}$, $\mathtt{bt(X)|pc(X)}$ etc. ○

Roughly speaking, CLAUDIEN works as follows (for a detailed discussion we refer to [De Raedt and Dehaspe, 1997]). It keeps track of a list of candidate clauses $Q$, which is initialized to the maximally general clause (with respect to the given syntactic bias $\mathcal{L}$). It repeatedly deletes a clause $c$ from $Q$, and tests whether $c$ is valid on the data. If it is, $c$ is added to the final hypothesis, otherwise, all maximally general specializations of $c$ (with respect to $\mathcal{L}$) are computed. and added back to $Q$. This process continues until $Q$ is empty and all relevant parts of the search space have been considered.

The clauses $\mathcal{C}$ generated by CLAUDIEN can be used to form an initial hypothesis. In the experiments presented below, for each predicate, we selected one of the clauses generated by CLAUDIEN for inclusion in the initial hypothesis such that the valid Bayesian logic program was also acyclic on the data cases (see below).

**Example 4.8** An initial hypothesis is e.g. the program $H_0$

```
mc(X)  | m(Y, X).
pc(X)  | f(Y, X).
bt(X)  | mc(X).
```

○

The second modification is concerned with pruning away those candidate Bayesian logic programs that are logically invalid. This is realized by the first **if**-condition in the loop. The second **if**-condition tests whether cyclic dependencies are induced on the data cases. This can be performed in time $O(s \cdot r^3)$ where $r$ is the number of random variables of the largest data case in $\mathbf{D}$ and $s$ is the number of clauses in $H$. To realize this, we build the Bayesian networks induced by $H$ over each $\mathrm{RandVar}(D_i)$ by computing the ground instances for each clause $c \in H$ for which the ground atoms are members of $\mathrm{RandVar}(D_i)$. Thus, ground atoms not appearing as a head atom of a valid ground instance, are apriori nodes, i.e. nodes with an empty parent set. This takes $O(s \cdot r_i^3)$. Then, we test in $O(r_i)$ whether a topological order of the nodes in the induced Bayesian network exists. If it does, the Bayesian network is acyclic.

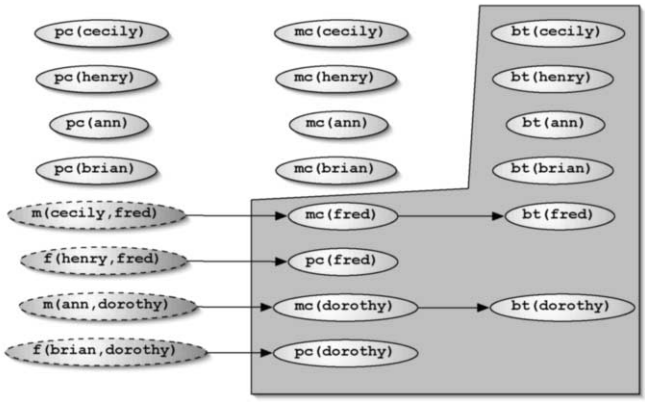



**Figure 4.2.** Support network induced by the initial hypothesis $H_0$ (see text) over the data cases $D_1$ and $D_2$. The gray region indicates random variables, which have been generated by the Bayesian clauses. Random variables out of the gray region have not been generated by the Bayesian clauses; they are asserted as extensional facts.

Otherwise, it is cyclic. In total, the run time is $O(m \cdot s \cdot r^3) = O(s \cdot r^3)$ because the given data set is fixed. Figure 4.2 shows the support network induced by the initial hypothesis on $D_1$ and $D_2$.

**Use of ILP Refinement Operators:** Difference (2) is the employment of refinement operators traditionally used in ILP, cf. Definition 2.13. It is a simple consequence of working with first order clauses instead of propositional one to encode dependencies. For the language bias considered in our experiments, we use the two refinement operators $\rho_s : 2^{\mathcal{H}} \mapsto \mathcal{H}$ and $\rho_g : 2^{\mathcal{H}} \mapsto \mathcal{H}$. The operator $\rho_s(H)$ adds constant-free atoms to the body of a single clause $c \in H$, and $\rho_g(H)$ deletes constant-free atoms from the body of a single clause $c \in H$. Other refinement operators such as deleting and adding logically valid clauses, instantiating variables, and unifying variables are possible, too, cf. [Nienhuys-Cheng and de Wolf, 1997]. Figure 4.1 shows the different refinement operators for the first order case and the propositional case for learning Bayesian networks. Instead of adding (deleting) propositions to (from) the body of a clause, they add (delete) according to our language assumption constant-free atoms. Consequently, they add multiple edges into or delete multiple edges from the underlying support network. Furthermore, Figure 4.1 shows that using the refinement operators each hypothesis (within our language bias) can — in principle — be reached.

Finally, we need to mention that although the maximally general clauses are the most interesting ones from the logical point of view, this is not necessarily the case from the probabilistic point of view.

**Example 4.9** For the data cases $D_1$ and $D_2$, the initial candidate $H_0$

```
mc(X)  | m(Y, X).
```

```
pc(X)  | f(Y, X).
bt(X)  | mc(X).
```

will probably not obtain the best possible score. The reason is that the blood type does not depend on the genetical information of the father, cf. Figure 4.2. ○

### 4.2.3 Discussion and Possible Extensions

The algorithm presented serves as a basic approach for learning Bayesian logic programs from data. Various extensions and modifications based on ideas developed either ILP or Bayesian networks learning, are possible and often straightforward. Obvious modifications include *tabu search*, where the $k$ recently visited hypothesis are avoided during search, *beam search*, where we keep track of the $k$ best hypotheses rather than just the best one, *random restarts* when getting stuck, and *simulated annealing*, where refinements are evaluated in random order [Russell and Norvig, 1995]. More interesting modifications are concerned with *lookahead*, *background knowledge*, and *improved scoring functions*. Let us briefly address the latter two ones.

We have discussed lookaheads and background knowledge already for ILP in Section 2.2. Lookaheads apply without modifications. Background knowledge for Bayesian logic programs consists of a fixed Bayesian logic program $B$. One can then search for the candidate $H^*$, which is together with $B$ acyclic on the data such that for all

$$D_i \in \mathbf{D} : \mathrm{LH}(B \cup H^* \cup \mathrm{RandVar}(D_i)) = \mathrm{RandVar}(D_i) \ ,$$

and $B \cup H^*$ scores best on the data $\mathbf{D}$ according to $\mathrm{score}_{\mathbf{D}}$. This is particularly interesting to specify deterministic knowledge as in ILP and to specify hidden variables as in learning Bayesian networks. Alternatively, one might also fix the structure of *some* of the clauses only, but not necessarily their parameters.

Using the likelihood directly as scoring function, score-based algorithms for learning Bayesian networks prefer fully connected networks. To overcome this problem, more advanced scoring functions have been developed. One of these is the *minimum description length* (MDL) score, which trades off the fit to the data with the complexity of the network. In the context of learning Bayesian networks, the whole Bayesian network is encoded to measure compression [Lam and Bacchus, 1994]. In the context of learning logic programs, other compression measures have been investigated such as the average length of proofs [Srinivasan et al., 1994]. For Bayesian logic programs, an integration of both measures seems appropriate.

Finally, we wish to stress that within Scooby one could also learn predicate definitions consisting of more than one clause. The refinement operator would then need to be also modified such that for a clause $c \in H'$ with head predicate $p$ another (valid) clause $c'$ (e.g. computed by Claudien) with head predicate $p$ may also be added or deleted. However, using these operations, the log-likelihood score is not an appropriate score anymore.

**Example 4.10** Recall the *fever* example 3.12 where `fever` depends probabilistically on `cold`, `flu`, and `malaria`. More precisely,`fever` is the *noisy-or* of `cold`, `flu`, and

`malaria`. We can represent this hypothesis in two different ways: first, as a single Bayesian clause $c$ where $cpd(c)$ encodes *noisy-or* and, second, as three clauses with *noisy-or* as combining rule (as done in Section 3.2.2). Both representations will yield the same likelihood. ◦

Again, MDL measures that give a penalty for longer description length of the hypothesis are promising candidates in this direction.

## 4.3 Parameter Estimation

So far, we have assumed that there is an algorithm that finds the optimal parameters of a candidate program given the data. In this section, we show how to learn this quantitative component, i.e. the conditional probability distributions, of Bayesian logic programs. The learning problem can be stated as follows

**Definition 4.11** (Parameter Estimation Problem) **Given** a set $\mathbf{D} = \{D_1, \ldots, D_m\}$ of data cases [14], a set $H$ of Bayesian clauses, which is logically valid and acyclic on the data, a combining rule for each Bayesian predicate, and a scoring function $\text{score}_{\mathbf{D}} : \mathcal{H} \mapsto \mathbb{R}$, **find** the parameter values of $H$ that maximize $\text{score}_{\mathbf{D}}$. ◦

We will concentrate on the classical *maximum likelihood estimation* (MLE) approach.

### 4.3.1 Maximum Likelihood Estimation

*Maximum likelihood* is a classical method for parameter estimation. The likelihood is the probability of the observed data as a function of the unknown parameters with respect to the current model. Let $B$ be a Bayesian logic program consisting of the Bayesian clauses $c_1, \ldots, c_n$, and let $\mathbf{D} = \{D_1, \ldots, D_m\}$ be a set of data cases. The parameters

$$\text{cpd}(c_i)_{jk} = P(u_j \mid \mathbf{u}_k),$$

where $u_j \in \text{D}(\text{head}(c_i))$ and $\mathbf{u}_k \in \text{D}(\text{body}(c_i))$, affecting the associated conditional probability distributions $\text{cpd}(c_i)$ constitute the set $\boldsymbol{\lambda} = \bigcup_{i=1}^{n} \text{cpd}(c_i)$. The version of $B$ where the parameters are set to $\boldsymbol{\lambda}$ is denoted by $B(\boldsymbol{\lambda})$, and as long as no ambiguities occur we will not distinguish between the parameters $\boldsymbol{\lambda}$ themselves and a particular instance of them.

Now, the likelihood $L(\mathbf{D}, \boldsymbol{\lambda})$ is the probability of the data $\mathbf{D}$ as a function of the unknown parameters $\boldsymbol{\lambda}$:

$$L(\mathbf{D}, \boldsymbol{\lambda}) := P_B(\mathbf{D} \mid \boldsymbol{\lambda}) = P_{B(\boldsymbol{\lambda})}(\mathbf{D}). \tag{4.1}$$

[14] Given a well-defined Bayesian network $B$, the logical part of a data case $D_i$ can also be a partial model only if we only estimate the parameters and do not learn the structure, i.e. $RandVar(D_i) \subseteq \text{LH}(B)$. The given Bayesian logic program will fill in the missing random variables.

Thus, the search space $\mathcal{H}$ is spanned by the product space over the possible values of $\lambda(c_i)$ and we seek to find the parameter values $\boldsymbol{\lambda}^*$ that maximize the likelihood, i.e.

$$\boldsymbol{\lambda}^* = \max_{\boldsymbol{\lambda} \in \mathcal{H}} P_{B(\boldsymbol{\lambda})}(\mathbf{D}).$$

Usually, $B$ specifies a distribution over a (countably) infinite set of random variables namely $\mathrm{LH}(B)$ and hence we cannot compute $P_{B(\boldsymbol{\lambda})}(\mathbf{D})$ by considering the whole dependency graph. But as we have argued in Section 3.2.3 it is sufficient to consider the support network (see Definition 3.16 $N(\boldsymbol{\lambda})$ of the random variables occurring in $\mathbf{D}$ to compute $P_{B(\boldsymbol{\lambda})}(\mathbf{D})$. Thus, using the monotonicity of the logarithm, we seek to find

$$\boldsymbol{\lambda}^* = \max_{\boldsymbol{\lambda} \in \mathcal{H}} \log P_{N(\boldsymbol{\lambda})}(\mathbf{D}) \tag{4.2}$$

where $P_{N(\boldsymbol{\lambda})}$ is the probability distribution specified by the support network $N(\boldsymbol{\lambda})$ of the random variables occurring in $\mathbf{D}$. Equation (4.2) expresses the original problem in terms of the maximum likelihood parameter estimation problem of Bayesian networks:

**Observation** A Bayesian logic program together with data cases induces a Bayesian network over the variables of the data cases.

This is not surprising because the learning setting is an instance of the probabilistic learning from interpretations, see Section 2.3.1. More important, due to the reduction, all techniques for maximum likelihood parameter estimation within Bayesian networks are in principle applicable. We only need to take the following issues into account:

(1) Some of the nodes in $N(\boldsymbol{\lambda})$ are hidden, i.e., their values are not observed in $\mathbf{D}$.
(2) We are not interested in the conditional probability distributions associated to ground instances of Bayesian clauses, but in those associated to the Bayesian clauses themselves.
(3) Not only $L(\mathbf{D}, \boldsymbol{\lambda})$ but also $N(\boldsymbol{\lambda})$ itself depends on the data, i.e. the data cases determine the subnetwork of $DG(B)$ that is sufficient to calculate the likelihood.

The available data cases may not be complete, i.e., some values may not be observed. For instance in medical domains, a patient rarely gets all of the possible tests. In presence of missing data, the maximum likelihood estimate typically cannot be written in closed form. Unfortunately, it is a numerical optimization problem, and all known algorithms involve nonlinear, iterative optimization and multiple calls to a Bayesian inference procedures as subroutines, which are typically computationally infeasible. For instance the inference within Bayesian network has been proven to be NP-hard [Cooper, 1990]. Typical ML parameter estimation techniques (in the presence of missing data) are the Expectation-Maximization (EM) algorithm and gradient-based approaches. We will now discuss both approaches in turn.

### 4.3.2 Gradient-based Approach

We will adapt Binder et al.'s [1997] solution for dynamic Bayesian networks based on the chain rule of differentiation. For simplicity, we fix the current instantiation of

the parameters $\boldsymbol{\lambda}$ and, hence, we write $B$ and $N(\mathbf{D})$. Applying the chain rule to (4.2) yields

$$\frac{\partial \log P_N(\mathbf{D})}{\partial \operatorname{cpd}(c_i)_{jk}} = \sum_{\substack{\text{subst. } \theta \text{ s.t.} \\ \text{support}(c_i\theta)}} \frac{\partial \log P_N(\mathbf{D})}{\partial \operatorname{cpd}(c_i\theta)_{jk}} \tag{4.3}$$

where $\theta$ refers to grounding substitutions and $\text{support}(c_i\theta)$ is true iff $\{\text{head}(c_i\theta)\} \cup \text{body}(c_i\theta) \subset N$. Assuming that the data cases $D_l \in \mathbf{D}$ are independently sampled from the same distribution we can separate the contribution of the different data cases to the partial derivative of a single ground instance $c\theta$:

$$\begin{aligned}
\frac{\partial \log P_N(\mathbf{D})}{\partial \operatorname{cpd}(c_i\theta)_{jk}} &= \frac{\partial \log \prod_{l=1}^{m} P_N(D_l)}{\partial \operatorname{cpd}(c_i\theta)_{jk}} && \text{by independence} \\
&= \sum_{l=1}^{m} \frac{\partial \log P_N(D_l)}{\partial \operatorname{cpd}(c_i\theta)_{jk}} && \text{by } \log\prod = \sum \log \\
&= \sum_{l=1}^{m} \frac{\partial P_N(D_l)/\partial \operatorname{cpd}(c_i\theta)_{jk}}{P_N(D_l)} \,. && (4.4)
\end{aligned}$$

In order to obtain computations local to the parameter $\operatorname{cpd}(c_i\theta)_{jk}$ we introduce the variables $\text{head}(c_i\theta)$ and $\text{body}(c_i\theta)$ into the numerator of the summand of (4.4) and average over their possible values, i.e.,

$$\frac{\partial P_N(D_l)}{\partial \operatorname{cpd}(c_i\theta)_{jk}} = \frac{\partial}{\partial \operatorname{cpd}(c_i\theta)_{jk}} \Bigg( \sum_{j',k'} P_N(D_l, \text{head}(c_i\theta) = u_{j'}, \text{body}(c_i\theta) = \mathbf{u}_{k'}) \Bigg)$$

Applying the chain rule yields

$$\begin{aligned}
\frac{\partial P_N(D_l)}{\partial \operatorname{cpd}(c_i\theta)_{jk}} &= \frac{\partial}{\partial \operatorname{cpd}(c_i\theta)_{jk}} \Bigg( \sum_{j',k'} P_N(D_l \mid \text{head}(c_i\theta) = u_{j'}, \text{body}(c_i\theta) = \mathbf{u}_{k'}) \\
&\qquad \cdot P_N(\text{head}(c_i\theta) = u_{j'}, \text{body}(c_i\theta) = \mathbf{u}_{k'}) \Bigg) \\
&= \frac{\partial}{\partial \operatorname{cpd}(c_i\theta)_{jk}} \Bigg( \sum_{j',k'} P_N(D_l \mid \text{head}(c_i\theta) = u_{j'}, \text{body}(c_i\theta) = \mathbf{u}_{k'}) \\
&\qquad \cdot P_N(\text{head}(c_i\theta) = u_{j'} \mid \text{body}(c_i\theta) = \mathbf{u}_{k'}) \\
&\qquad \cdot P_N(\text{body}(c_i\theta) = \mathbf{u}_{k'}) \Bigg) \qquad (4.5)
\end{aligned}$$

where $u_j \in \text{D}(\text{head}(c_i))$, $\mathbf{u}_k \in \text{D}(\text{body}(c_i))$ and $j, k$ refer to the corresponding entries in $\operatorname{cpd}(c_i)$, respectively $\operatorname{cpd}(c_i\theta)$. In (4.5), $\operatorname{cpd}(c_i\theta)_{jk}$ appears only in linear form. Moreover, it appears only when $j' = j$, and $k' = k$. Therefore, (4.5) simplifies two

$$\frac{\partial P_N(D_l)}{\partial \operatorname{cpd}(c_i\theta)_{jk}} = P_N(D_l \mid \text{head}(c_i\theta) = u_j, \text{body}(c_i\theta) = \mathbf{u}_k) \cdot P_N(\text{body}(c_i\theta) = \mathbf{u}_k). \tag{4.6}$$

Substituting (4.6) back into (4.4) yields

$$\begin{aligned}
&\sum_{l=1}^{m} \frac{\partial \log P_N(D_l)/\partial \operatorname{cpd}(c_i\theta)_{jk}}{P_N(D_l)} \\
&= \sum_{l=1}^{m} \frac{P_N(D_l \mid \operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k) \cdot P_N(\operatorname{body}(c_i\theta) = \mathbf{u}_k)}{P_N(D_l)} \\
&= \sum_{l=1}^{m} \frac{P_N(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k \mid D_l) \cdot P_N(D_l) \cdot P_N(\operatorname{body}(c_i\theta) = \mathbf{u}_k)}{P_N(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k) \cdot P_N(D_l)} \\
&= \sum_{l=1}^{m} \frac{P_N(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k \mid D_l)}{P_N(\operatorname{head}(c_i\theta) = u_j \mid \operatorname{body}(c_i\theta) = \mathbf{u}_k)} \\
&= \sum_{l=1}^{m} \frac{P_N(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k \mid D_l)}{\operatorname{cpd}(c_i\theta)_{jk}} \,.
\end{aligned}$$

Combining all these, (4.3) can be rewritten as

$$\frac{\partial \log P_N(\mathbf{D})}{\partial \operatorname{cpd}(c_i)_{jk}} = \sum_{\substack{\text{subst. } \theta \text{ with} \\ \text{support}(c_i\theta)}} \frac{\operatorname{en}(c_{ijk} \mid \theta, \mathbf{D})}{\operatorname{cpd}(c_i\theta)_{jk}} \tag{4.7}$$

where

$$\begin{aligned}
\operatorname{en}(c_{ijk} \mid \theta, \mathbf{D}) &:= \operatorname{en}(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k \mid \mathbf{D}) \\
&:= \sum_{l=1}^{m} P_N(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k \mid D_l)
\end{aligned} \tag{4.8}$$

are the so-called *expected counts* of the joint state $\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k$ given the data $\mathbf{D}$.

Equation (4.7) shows that

$$P_N(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k \mid D_l)$$

is all what is needed. This can essentially be computed using any standard Bayesian network inference engine. This is not surprising because (4.7) differs from the one for Bayesian networks given in [Binder et al., 1997] only in that we sum over all ground instances of a Bayesian clause holding in the data. To stress this close relationship, we rewrite (4.7) in terms of expected counts of clauses instead of ground clauses. They are defined as follows:

**Definition 4.12** (Expected Counts of Bayesian Clauses) The expected counts of a Bayesian clauses $c$ of a Bayesian logic program $B$ for a data set $\mathbf{D}$ are defined as

$$\begin{aligned}
\operatorname{en}(c_{ijk} \mid \mathbf{D}) &:= \operatorname{en}(\operatorname{head}(c_i) = u_j, \operatorname{body}(c_i) = \mathbf{u}_k \mid \mathbf{D}) \\
&:= \sum_{\substack{\text{subst. } \theta \text{ with} \\ \text{support}(c_i\theta)}} \operatorname{en}(\operatorname{head}(c_i\theta) = u_j, \operatorname{body}(c_i\theta) = \mathbf{u}_k \mid \mathbf{D}) \,.
\end{aligned} \tag{4.9}$$

○

**Algorithm II.2**: A simplified skeleton of the algorithm for *adaptive Bayesian logic programs* estimating the parameters of a Bayesian logic program.

**input** : $B$, a Bayesian logic program; associated cpds are parameterized by $\boldsymbol{\lambda}$; $\mathbf{D}$, a finite set of data cases
**output**: a modified Bayesian logic program

```
λ ← InitialParameters
N ← SupportNetwork(B, D)
repeat
    Δλ ← 0
    set associated conditional probability distribution of N according to λ
    foreach D_l ∈ D do
        set the evidence in N from D_l
        foreach Bayesian clause c ∈ B do
            foreach ground instance cθ s.t. {head(cθ)} ∪ body(cθ) ⊂ N do
                foreach single parameter cpd(cθ)_jk do
                    Δ cpd(c)_jk ← Δ cpd(c)_jk + (∂ log P_N(D_l) / ∂ cpd(cθ)_jk)
    Δλ ← ProjectionOntoConstraintSurface(Δλ)
    λ ← λ + α · Δλ
until Δλ ≈ 0
return B
```

Reading (4.7) in terms of definition 4.12 proves the following proposition:

**Proposition 4.13** (Partial Derivative of Log-Likelihood) *Let $B$ be a Bayesian logic program with parameter vector $\boldsymbol{\lambda}$. The partial derivative of the log-likelihood of $B$ with respect to $\mathrm{cpd}(c_i)_{jk}$ for a given data set $\mathbf{D}$ is*

$$\frac{\partial LL(\mathbf{D}, \boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_i)_{jk}} = \frac{\mathrm{en}(c_{ijk} \mid \mathbf{D})}{\mathrm{cpd}(c_i)_{jk}} \,. \tag{4.10}$$

∘

Equation (4.10) can be viewed as the first-order logical equivalent of the Bayesian network formula. A simplified skeleton of a gradient-based algorithm employing (4.10) is shown in Algorithm II.2.

Before showing how to adapt the EM algorithm, we have to explain two points, which we have left out so far for the sake of simplicity: Constraint satisfaction and decomposable combining rules.

In the problem at hand, the gradient ascent has to be modified to take into account the constraint that the parameter vector $\boldsymbol{\lambda}$ consists of probability values, i.e. $\mathrm{cpd}(c_i)_{jk} \in [0, 1]$ and $\sum_j \mathrm{cpd}(c_i)_{jk} = 1$ . Following Binder et al. [1997], there are two ways to enforce this:

(1) Projecting the gradient onto the constraint surface (as used to formulate the Algorithm II.2), and
(2) reparameterizing the problem.

In the experiments, we chose the reparameterization approach because the new parameters automatically respect the constraints on $\mathrm{cpd}(c_i)_{jk}$ no matter what their values are. More precisely, we define the parameters $\boldsymbol{\beta}$ with $\beta_{ijk} \in \mathbb{R}$ such that

$$\mathrm{cpd}(c_i)_{jk} = \frac{e^{\beta_{ijk}}}{\sum_l e^{\beta_{ilk}}} \tag{4.11}$$

where the $\beta_{ijk}$ are indexed like $\mathrm{cpd}(c_i)_{jk}$. This enforces the constraints given above, and a local maximum with respect to the $\boldsymbol{\beta}$ is also a local maximum with respect to $\boldsymbol{\lambda}$, and vice versa. The gradient with respect to $\boldsymbol{\beta}$ can be found by computing the gradient with respect to $\boldsymbol{\lambda}$ and then deriving the gradient with respect to $\boldsymbol{\beta}$ using the chain rule of derivatives. More precisely, the chain rule of derivatives yields

$$\frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \beta_{ijk}} = \sum_{i'j'k'} \frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_{i'})_{j'k'}} \cdot \frac{\partial \,\mathrm{cpd}(c_{i'})_{j'k'}}{\partial \beta_{ijk}} \tag{4.12}$$

Since $\partial \,\mathrm{cpd}(c_{i'})_{j'k'}/\partial \beta_{ijk} = 0$ for all $i \neq i'$, and $k \neq k'$, (4.12) simplifies to

$$\frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \beta_{ijk}} = \sum_{j'} \frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_i)_{j'k}} \cdot \frac{\partial \,\mathrm{cpd}(c_i)_{j'k}}{\partial \beta_{ijk}}$$

The quotient rule yields

$$\begin{aligned}\frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \beta_{ijk}} &= \sum_{j'} \left\{ \frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_i)_{j'k}} \cdot \frac{\left(\frac{\partial e^{\beta_{ij'k}}}{\partial \beta_{ijk}} \cdot \sum_l e^{\beta_{ilk}}\right) - \left(e^{\beta_{ij'k}} \cdot \frac{\partial \sum_l e^{\beta_{ilk}}}{\partial \beta_{ijk}}\right)}{\left(\sum_l e^{\beta_{ilk}}\right)^2} \right\} \\ &= \frac{\left\{ \sum_{j'} \left( \frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_i)_{j'k}} \cdot \frac{\partial e^{\beta_{ij'k}}}{\partial \beta_{ijk}} \cdot \sum_l e^{\beta_{ilk}} \right) \right\}}{\left(\sum_l e^{\beta_{ilk}}\right)^2} - \frac{\left\{ \sum_{j'} \left( \frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_i)_{j'k}} \cdot e^{\beta_{ij'k}} \cdot \frac{\partial \sum_l e^{\beta_{ilk}}}{\partial \beta_{ijk}} \right) \right\}}{\left(\sum_l e^{\beta_{ilk}}\right)^2}\end{aligned}$$

Because $\partial e^{\beta_{ij'k}}/\partial \beta_{ijk} = 0$ for $j' \neq j$ and $\partial e^{\beta_{ijk}}/\partial \beta_{ijk} = e^{\beta}_{ijk}$, this simplifies to

$$\frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \beta_{ijk}} = \frac{\left( \frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_i)_{jk}} \cdot e^{\beta_{ijk}} \cdot \sum_l e^{\beta_{ilk}} \right)}{\left(\sum_l e^{\beta_{ilk}}\right)^2} - \frac{\sum_{j'} \left( \frac{\partial LL(\mathbf{D},\boldsymbol{\lambda})}{\partial \,\mathrm{cpd}(c_i)_{j'k}} \cdot e^{\beta_{ij'k}} \cdot e^{\beta_{ilk}} \right)}{\left(\sum_l e^{\beta_{ilk}}\right)^2}$$

$$= \frac{e^{\beta_{ijk}}}{\left(\sum_l e^{\beta_{ilk}}\right)^2} \cdot \left\{ \frac{\partial LL(\mathbf{D}, \boldsymbol{\lambda})}{\partial \operatorname{cpd}(c_i)_{jk}} \cdot \left(\sum\nolimits_l e^{\beta_{ilk}}\right) - \sum\nolimits_{j'} \frac{\partial LL(\mathbf{D}, \boldsymbol{\lambda})}{\partial \operatorname{cpd}(c_i)_{j'k}} \cdot e^{\beta_{ij'k}} \right\} \tag{4.13}$$

To further simplify the partial derivative, we note that $\partial LL(\mathbf{D}, \boldsymbol{\lambda}) / \partial \operatorname{cpd}(c_i)_{jk}$ can be rewritten as

$$\frac{\partial LL(\mathbf{D}, \boldsymbol{\lambda}}{\partial \operatorname{cpd}(c_i)_{jk}} = \frac{\operatorname{en}(c_{ijk} \mid \mathbf{D})}{\operatorname{cpd}(c_i)_{jk}} = \frac{\operatorname{en}(c_{ijk} \mid \mathbf{D})}{\frac{e^{ijk}}{\sum_l e^{\beta_{ijk}}}} = \frac{\operatorname{en}(c_{ijk} \mid \mathbf{D})}{e^{\beta_{ijk}}} \cdot \left(\sum\nolimits_l e^{\beta_{ijk}}\right)$$

by substituting (4.11) in (4.9). Using the last equation, (4.13) simplifies to

$$\begin{aligned} &\frac{\partial LL(\mathbf{D}, \boldsymbol{\lambda})}{\partial \beta_{ijk}} \\ &= \frac{e^{\beta_{ijk}}}{\left(\sum_l e^{\beta_{ilk}}\right)^2} \cdot \left\{ \frac{\operatorname{en}(c_{ijk} \mid \mathbf{D})}{e^{\beta_{ijk}}} \cdot \left(\sum\nolimits_l e^{\beta_{ilk}}\right)^2 - \sum\nolimits_{j'} \frac{\operatorname{en}(c_{ij'k} \mid \mathbf{D})}{e^{\beta_{ij'k}}} \cdot \left(\sum\nolimits_l e^{\beta_{ilk}}\right) \cdot e^{\beta_{ij'k}} \right\} \\ &= \operatorname{en}(c_{ijk} \mid \mathbf{D}) - \frac{e^{\beta_{ijk}}}{\sum_l e^{\beta_{ilk}}} \cdot \sum\nolimits_{j'} \operatorname{en}(c_{ij'k} \mid \mathbf{D}) \quad . \end{aligned}$$

Using once more (4.11), the following proposition is proven:

**Proposition 4.14** (Partial Derivative of Log-Likelihood of an Reparameterized BLP) *Let $B$ be a Bayesian logic program reparameterized according to* (4.11)*. The partial derivative of the log-likelihood of $B$ with respect to $\beta_{ijk}$ for a given data set* $\mathbf{D}$ *is*

$$\frac{\partial LL(\mathbf{D}, \boldsymbol{\lambda})}{\partial \beta_{ijk}} = \operatorname{en}(c_{ijk} \mid \mathbf{D}) - \operatorname{cpd}(c_i)_{jk} \sum\nolimits_{j'} \operatorname{en}(c_{ij'k} \mid \mathbf{D}) \; . \tag{4.14}$$

∘

Equation (4.14) shows that the partial derivative can be expressed solely in terms of expected counts and original parameters. Consequently, its computational complexity is linear in (4.10).

We assumed *decomposable* combining rules.

**Definition 4.15** (Decomposable Combining Rule) Decomposable combining rules can be expressed using a set of separate, deterministic nodes in the support network such that the family of every non-deterministic node uniquely corresponds to a ground Bayesian clause, as shown in Figure 4.3. ∘

Most combining rules commonly employed in Bayesian networks such as *noisy_or* or linear regression are decomposable (cp. [Heckerman and Breese, 1994]). The definition of decomposable combining rules directly imply the following proposition.

**Proposition 4.16** *For each node $x$ in the support network $n$ there exist at most one clause $c$ and a substitution $\theta$ such that* $\operatorname{body}(c\theta) \subset \operatorname{LH}(B)$ *and* $\operatorname{head}(c\theta) = x$. ∘

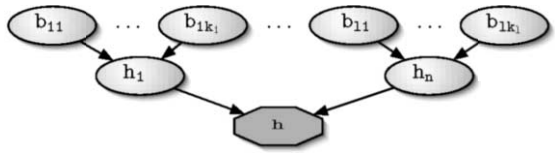



**Figure 4.3.** The scheme of decomposable combining rules. Each rectangle corresponds to a ground instance $c\theta \equiv \mathtt{h_i}|\mathtt{b_{1i}}, \ldots, \mathtt{b_{ki}}$ of a Bayesian clause $c \equiv \mathtt{h}|\mathtt{b_1}, \ldots, \mathtt{b_k}$. The node `h` is a deterministic node, i.e., its state is deterministic function of the parents joint state.

Thus, while the same clause $c$ can induce more than one node in $N$, all of these nodes have identical local structure: the associated conditional probability distributions (and so the parameters) have to be identical, i.e.,

$$\forall \text{ substitutions } \theta: \ \mathrm{cpd}(c\theta) = \mathrm{cpd}(c) \ .$$

**Example 4.17** Consider the nodes `bt(ann)`, `mc(ann)`, `pc(ann)` and `bt(brain)`, `mc(brain)`, `pc(brian)`. Both families contribute to the conditional probability distribution associated with the clause defining `bt(X)`. ○

This is the same situation as for dynamic Bayesian networks where the parameters that encode the stochastic model of state evolution appear many times in the network. However, gradient methods might be applied to non-decomposable combining functions as well. In the general case, the partial derivatives of an inner function has to be computed. For instance, Binder et al. [1997] derive the gradient for *noisy_or* when it is not expressed in the structure. This seems to be more difficult in the case of the EM algorithm, which we will now devise.

### 4.3.3 Expectation-Maximization (EM)

The Expectation-Maximization (EM) algorithm [Dempster et al., 1977] is another classical approach to maximum likelihood parameter estimation in the presence of missing values. The basic observation of the Expectation-Maximization algorithm is as that

*if the states of all random variables are observed, then learning would be easy.*

Assuming that no value is missing, Lauritzen [1995] showed that maximum likelihood estimation of Bayesian network parameters simply corresponds to frequency counting in the following way. Let $n(\mathbf{a} \mid \mathbf{D})$ denote the *counts* for a particular joint state $\mathbf{a}$ of variables $\mathbf{A}$ in the data, i.e. the number of cases in which the variables in $\mathbf{A}$ are assigned the evidence $\mathbf{a}$. Then the maximum likelihood value for the conditional probability value $P(X = x | \mathbf{Pa}(X) = \mathbf{u})$ is the ratio

$$\frac{n(X = x, \mathbf{Pa}(X) = \mathbf{u}_k \mid D_l)}{n(\mathbf{Pa}(X) = \mathbf{u}_k \mid D_l)} \ . \tag{4.15}$$

However, in the presence of missing values, the maximum likelihood estimates typically cannot be written in closed form. Therefore, the Expectation-Maximization algorithm iteratively performs the following two steps:

**(E-Step)** Based on the current parameters $\boldsymbol{\lambda}$ and the observed data $\mathbf{D}$ the algorithm computes a distribution over all possible completions of each partially observed data case. Each completion is then treated as a fully-observed data case weighted by its probability.

**(M-Step)** A new set of parameters is then computed based on Equation (4.15) taking the weights into accounts.

Lauritzen [1995] showed that this idea leads to a modified Equation (4.15) where the *expected counts*

$$\mathrm{en}(\mathbf{a}|\mathbf{D}) := \sum_{l=1}^{m} P_N(\mathbf{a} \mid D_l) \tag{4.16}$$

are used instead of *counts*. Again, essentially any Bayesian network engine can be used to compute $P(\mathbf{a}|D_l)$.

To apply the EM algorithm to parameter estimation of Bayesian logic programs, we assume decomposable combining rules. Thus,

- each node in the support network was "produced" by exactly one Bayesian clause $c$, and
- each node derived from $c$ can be seen as a separate "experiment" for the conditional probability distribution $\mathrm{cpd}(c)$.

Formally, due to the reduction of our problem at hand to parameter estimation within the support network $N$, the update rule becomes

$$\mathit{cpd}(c_i)_{jk} \leftarrow \frac{\mathrm{en}(c_i|\mathbf{D})}{\mathrm{en}(\mathrm{body}(c_i)|\mathbf{D})} = \frac{\mathrm{en}(\mathrm{head}(c_i), \mathrm{body}(c_i)|\mathbf{D})}{\mathrm{en}(\mathrm{body}(c_i)|\mathbf{D})} \tag{4.17}$$

where $\mathrm{en}(\cdot|\mathbf{D})$ refers to the first order expected counts as defined in Equation (4.9). Note that the summation over data cases and ground instances is hidden in $\mathrm{en}(\cdot|\mathbf{D})$. Equation (4.17) is similar to the one already encountered in Equation (4.10) for computing the gradient.

### 4.3.4 Gradient vs. EM

As one can see, the EM update rule in equation (4.17) and the corresponding equation (4.7) for the gradient ascent are very similar. Both rely on computing expected counts. They differ mainly in the way the expected counts are weighted. When the estimated probability is zero, this will dominate the evaluation of the Bayesian logic program. To avoid this difficulty one can adopt solutions to this problem well-known for the propositional case such as $m$-estimates (see e.g. [Mitchell, 1997]) or BDeu priors [Heckerman et al., 1995a]. However, gradient-based methods should be less

sensitive to informative priors than EM, because the EM uses the current parameters "only" to compute the expected counts, whereas gradient-based methods weight the expected counts by the current parameters. Thus, gradient-based approaches update the parameters in a more conservative manner by propagating the initial set of parameters through the iterations.

The comparison between EM and (advanced) gradient techniques like conjugate gradient is not yet well understood. Both methods perform a greedy local search, which is guaranteed to converge to stationary points. They both exploit expected counts, i.e., sufficient statistics as their primary computation step. However, there are important differences.

The EM is easier to implement because it does not have to enforce the constraint that the parameters are probability distributions. It converges much faster than simple gradient, and is somewhat less sensitive to starting points. (Conjugate) gradients estimate the step size (see below) with a line search involving several additional Bayesian network inferences compared to EM.

On the other hand, gradients are more flexible than EM, as they allow one to learn non-multinomial parameterizations using the chain rule for derivatives [Binder et al., 1997] or to choose other scoring functions than the likelihood [Jensen, 1999].

Furthermore, although the EM algorithm is quite successful in practice due to its simplicity and fast initial progress, it has been argued (see e.g. [Jamshidian and Jennrich, 1997, McLachlan and Krishnan, 1997] and references in there) that the EM convergence can be extremely slow, and that more advanced second-order methods should in general be favored to EM. In the context of Bayesian networks,Thiesson [1995], Bauer et al. [1997], and Ortiz and Kaelbling [1999a] investigated acceleration of the EM algorithm. All approaches rely on conventional (gradient-based) optimization techniques viewing the change in values in the parameters at an EM iteration as generalized gradient [15] (see [Ortiz and Kaelbling, 1999b] for a nice overview). Gradient ascent yields *parameterized EM*, and conjugate gradient yields *conjugate gradient EM* (CGEM). Although accelerated EMs can be significantly faster than EM, they all require more computational efforts than the basic EM. One reason is that they perform in each iteration a line search to choose an optimal step size. There are drawbacks of doing a line search. First, a line search introduces new problem-dependent parameters such as stopping criterion. Second, the line search involves several likelihood evaluations, which are NP-hard for Bayesian networks. Thus, the line search dominates the computational costs resulting in a disadvantage of the accelerated EM compared to the EM, which does one likelihood evaluation per iteration. The computational extra costs have to be amortized over the long run to gain a speed-up. Therefore, we we introduced in [Fischer and Kersting, 2003] a novel acceleration of EM called *scaled CGEM* (SCGEM), which overcomes the expensive line search. It evaluates the likelihood as often as the EM per iteration, namely once. SCGEM adopt the ideas underlying *scaled conjugate gradients* (SCGs), which are well-known from the field of learning neural networks [Møller, 1993]. SCG employs an approximation of the Hessian of the scoring function to quadratically extrapolate the minimum instead of doing a line search. Then, a Levenberg-Marquardt approach [Luenberger, 1984] scales

[15] Generalized gradients perform regular gradient techniques in a transformed space.

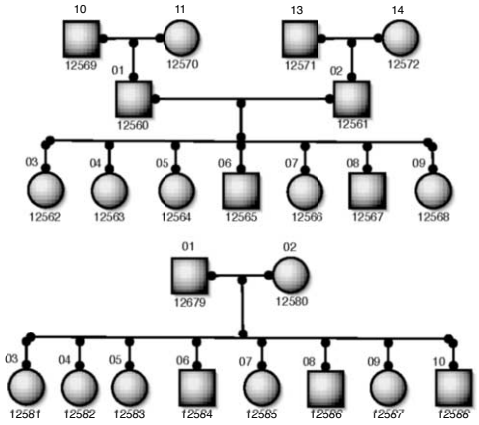



**Figure 4.4.** CEPH/French 12 (upper pedigree) and CEPH/French 17 (lower pedigree). Boxes denote males, circles denote females. A small number is the person's id within the pedigree, whereas a large number is the person's global id within the CEPH database.

the step size. SCGEM adopts the scaling mechanism for maximization and applies it to the expected information matrix. We refer to [Fischer and Kersting, 2003] for more details.

Finally, though we focused here on parameter estimation, methods for computing the gradient of the log-likelihood with respect to the parameters of a probabilistic model can also be used to employ generative models within discriminative learners such as SVMs. In the context of probabilistic ILP, this yields relational kernel methods, which we will introduce in Section 8.

## 4.4 Experimental Evaluation

Scooby is intended as a generic framework for learning Bayesian logic programs. As such, it leaves several issues open. These include: the actual language bias (and corresponding refinement operator), the scoring function, possible pruning methods, etc. Nevertheless, in order to show the validity of our framework, we report on some experiments that prove that the principles underlying Scooby work.

More specifically, we implemented Scooby in Sicstus Prolog 3.9.0. The experiments were run on a Pentium-IV 2.8 GHz Linux machine with 2.1 GB main memory. The implementation features a beam-search using the log-likelihood of the data as score, lookaheads, simple mode declarations, and the specification of extensional background knowledge. The chosen syntactic language bias is $\mathcal{L}_1$, cf. end of Section 4.1. The implementation has an interface to the Netica API (`http://www.norsys.com`) for Bayesian network inference and maximum likelihood estimation. To perform the maximum likelihood estimation, we implemented the EM algorithm and adapted the scaled conjugate gradient (SCG) as implemented in Bishop and Nabney's Netlab

| # beam size | 1 | | | 2 | | | 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| # data cases | 20 | 30 | 50 | 20 | 30 | 50 | 20 | 30 | 50 |
| miss rate 0.0 | - | - | + | + | + | + | + | + | + |
| miss rate 0.2 | - | - | - | - | + | + | + | + | + |

**Table 4.1.** The results of the *blood type* experiments. A '+' indicates that SCOOBY scored the original, intensional Bayesian clauses of the *bloodtype* program best; a '-' indicates that SCOOBY learned other clauses.

library (`http://www.ncrg.aston.ac.uk/netlab/`, see also [Bishop, 1995]) with an upper bound on the scale parameter of $2 \cdot 10^6$. For each score evaluation, the parameters were initialized randomly. To avoid zero entries in the conditional probability tables, $m$-estimates ($m = 1$) were used, see [Cestnik, 1990, Mitchell, 1997]. We used a simple, typical stopping criterion, which stopped when a change in log-likelihood was less than $10^{-3}$ from one iteration to the next.

Two experiments focusing on a different aspect of Bayesian logic programs, were performed. The first experiment was concerned with the question whether our SCOOBY could learn the structure of the *bloodtype* program from examples. The second experiment shows how SCOOBY can learn a probabilistic concept for the Bongard problem domain that is frequently used in ILP. Applications of Bayesian logic programs to important real-life data sets such as the KDD Cup 2001 data set on protein localization and web-page classification can be found in Chapter 8.3 on relational Fisher kernels.

### 4.4.1 Genetic Domain

The goal in the first experiment was to learn a global, descriptive model for our genetic domain, i.e. to learn the intensional Bayesian clauses of the *bloodtype* program. To do so, we used the CEPH (Centre d'Etude du Plymorphisme Humain) pedigrees[16] [Dausset et al., 1990], version 9.0, of all Amish, French and Venezuelan families (CEPH family numbers: $2, 12, 17, 21, 23, 28, 35, 37, 45, 66, 102, 104, 884$). These CEPH pedigrees are large and nuclear, i.e., they consist of two parents and their common offsprings. However, most of the CEPH pedigrees include all four associated grandparents, too. Figure 4.4 shows two CEPH pedigrees. We assumed all pedigrees to be independent, and extracted all structurally different pedigrees. This yielded 7 structurally different pedigrees. These 7 pedigrees were encoded together in one *family* (with 7 different components). Furthermore, the predicates `m/2` and `f/2` were declared to be *logical*. Recall from the end of Section 3.2.2 that logical predicates are specified by explicitly listing all true ground facts. The advantage is that one does not need to to consider the associated conditional probability distributions for such predicates. Lookahead declarations were used only for the predicates `m/2` and `f/2`. These specified that when an atom for these predicates was added, say `m(X,Y)`, one should also consider the refinements `m(X,Y), p(X)` and `m(X,Y), p(Y)` for all possible

[16] A pedigree is a record of a line of ancestors.

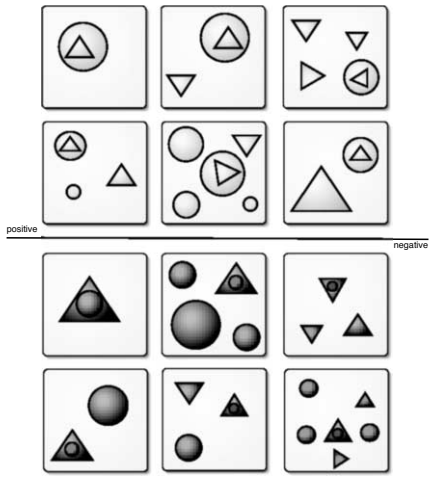



**Figure 4.5.** A Bongard problem consisting of 12 scenes, six positive ones and six negative ones. The goal is to discriminate between the two classes.

predicates `p/1`. Finally, with each Bayesian predicate, the *average* combining rule was associated.

We randomly sampled 20, 30, and 50 data cases (each representing a *family*) with both 0.0 and 0.2 values missing at random. This simulates 140, 210 and 350 (respectively) independent pedigrees for both miss rates because each independent family consists of 7 pedigrees. Then, we ran SCOOBY on each data case with beam sizes $1, 2$, and 3. The results are summarized in Table 4.1 where a + indicates that SCOOBY scored the original, intensional Bayesian clauses of the *bloodtype* program best; a - indicates that SCOOBY learned other clauses. As one can see, when the beam size and the number of data cases used is large enough, SCOOBY is able to learn all original clauses. In all negative cases, SCOOBY re-discovers

```
pc(X) | f(X,Y), mc(Y), pc(Y).
bt(X) | mc(X), pc(X).
```

but fails to find the correct definition of `mc/2`. It learned either `mc(X)|pc(X)` or `mc(X)|pc(X),father(Y,X),mc(Y)`. Thus, the first experiment clearly shows that SCOOBY is able to induce the structure of some simple Bayesian logic programs from data.

### 4.4.2 Bongard Domain

The Bongard problems (due to the Russian scientist M. Bongard) are well-known problems within ILP, cf. [Van Laer and De Raedt, 2001]. Consider Figure 4.5. Each example or scene consists of

| | # data cases | |
|---|---|---|
| miss rate | 100 | 300 |
| 0.0 | + | + |
| 0.2 | - | + |

**Table 4.2.** Further results of the *Bongards* experiments. A '+' indicates that SCOOBY scored the original, intensional Bayesian clauses best; a '-' indicates that SCOOBY learned other clauses.

- a variable number of geometrical objects such as triangles, rectangles and circles etc. (predicate `obj/2` with $\mathrm{D}(obj) = \{triangle, circle\}$), each having a number of different properties such as color, orientation, size etc., and
- a variable number of relations between objects such as *in* (predicate `in/2` having states *true*, *false*), *leftof*, *above* etc.

The usual task that has been addressed in the context of the Bongard problems within the field of ILP is that of classification, i.e. that of finding a set of rules, which *discriminates* the positive from the negative examples (represented by `class/1` over the states *pos*, *neg*). The popularity of the Bongard problems as a benchmark for ILP can be explained by the fact that they are very similar to real-world problems in e.g., the field of computational chemistry, where essentially the same representational problems arise. Indeed, in these problems, examples are molecules and are composed of several atoms with specific properties. Furthermore, there exist relations among the atoms, such as the bonds and possibly functional groups.

The purpose of our second experiment is to show that SCOOBY can learn probabilistic concepts from examples. A probabilistic concept is one that assigns a probability distribution to the possible classes. So, instead of classifying examples as positive or negative, a probabilistic concept would assign a probability of $p$ to the positive class and of 1-$p$ to the negative one. Notice that probabilistic concept-learning is beyond the scope of traditional ILP systems. A further difficulty with traditional ILP arises in the case of missing values. The equivalent of missing values in the usual ILP setting is that some of the facts describing the examples have an unknown truth-value, a problem that has not received much attention in the ILP literature.

The experiment was set up as follows. We randomly sampled 100 and 300 data cases with miss rates of 0.0, and 0.2. Each data set consisted of equally many scenes of 2, 4, 6 or 8 objects. An object was with probability 0.3 a circle and with probability 0.7 a triangle. We assumed the objects to be ordered, i.e., object `o1` can only be in `o2`, `o2` can only be in `o3`, etc. The probability of each single object to be in another object was 0.5. The probabilistic concept to be learned is specified by the following Bayesian clause:

| `obj(E,A)` | `in(A,B)` | `obj(E,B)` | $\mathbf{P}$(`class(E)`) |
|---|---|---|---|
| *triangle* | *true* | *triangle* | $(0.7, 0.3)$ |
| *triangle* | *true* | *circle* | $(0.7, 0.3)$ |
| ... | ... | ... | ... |
| *triangle* | *false* | *triangle* | $(0.1, 0.9)$ |
| ... | ... | ... | ... |

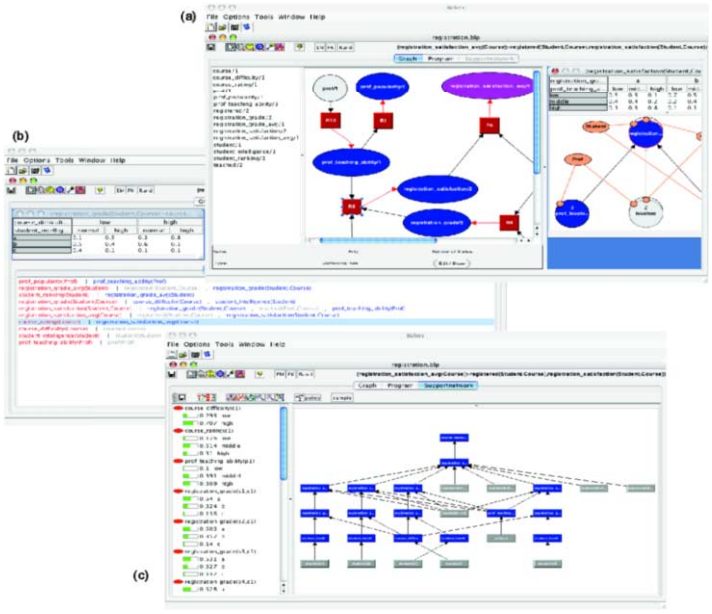



**Figure 4.6.** BALIOS. **(a)** Graphical representation of the *university* Bayesian logic program. **(b)** Textual representation of Bayesian clauses with associated conditional probability distributions. **(c)** Computed support network and probabilities for a probabilistic query.

where *noisy_or* is associated with `class/1`. The class of each example in the data set was then sampled from the probability distribution that the above Bayesian logic program assigns to each of the two classes.

We ran SCOOBY on each data set with beam size 1. We associated *noisy or* with all predicates. The `obj/2` predicate was declared to be logical. The results are summarized in Table 4.2, where + indicates that the structure of the learned Bayesian logic program was correct. Thus, this experiment shows that there exist tasks (such as probabilistic concept-learning) that are difficult (or uncommon) for traditional ILP systems but that can be addressed with SCOOBY and Bayesian logic programs.

## Balios – The Engine for Bayesian Logic Programs*

An engine for Bayesian logic programs featuring a graphical representation, logical atoms, and aggregate functions has been implemented in the BALIOS

* Builds on [Kersting and Dick, 2004].

system [Kersting and Dick, 2004], which is freely available for academic at `http://www.informatik.uni-freiburg.de/~kersting/achilleus/`.

Balios is written in Java. It calls Sicstus Prolog to perform logical inference and a Bayesian network inference engine (e.g. Hugin or Elvira) to perform probabilistic inference. Balios features a GUI graphically representing Bayesian logic programs, see Figure 4.6, computing the most likely configuration, exact (junction tree) and approximative inference methods (rejection, likelihood and Gibbs sampling), as well as parameter estimation methods (hard EM, EM and conjugate gradient).

# Future Work

With combining rules and aggregate functions, we offered two ways for handling multiple firing Bayesian clauses. Identifying cases, in which to prefer one method over the other one, remains an open question. One attractive alternative is to leave this question to the learner. For instance, Uwents and Blockeel's [2005] initiated research on relational neural networks to learn non-trivial combinations of aggregation and combining rules. Another alternative are non-parameteric approaches such as relational probability estimation trees [Fierens et al., 2005].

Trees could also be used to represent the conditional probability distributions associated with Bayesian clauses. We have only considered the naïve representation form, namely a tabular representation. This representation is exponential in the number of parents of a variable. As for Bayesian networks, this is a major problem when learning because the large numbers of parameters require large numbers of data cases to be assessed reliably. This is the reason why learning methods usually prefer model structures involving few parameters. Instead of penalizing the likelihood for complex model structures, one can also explicitly represent the local structure of conditional probability distributions, for instance using decision trees. This has been done for probabilistic relational models [Getoor, 2001].

One practical issue of the structure learning of Bayesian logic programs as presented in this thesis is its computational cost. To evaluate a single neighbour, the EM algorithm (respectively a gradient-based parameter estimation algorithm) has to run for several iterations in order to get reasonable expected counts. Each iteration requires a full Bayesian network inference on all data cases. Neighbours, however, differ from the current best hypothesis only in one Bayesian clause. Most parts of the program remain unchanged. In turn, it could be expected that the expected counts do not change dramatically as well. This idea has been followed within the structural EM (SEM) for learning the structure of Bayesian networks [Friedman, 1997]. SEM takes the current model $H_k$ and runs a parameter estimation algorithm for a while to get reasonably completed data. It then fixes the completed data cases and uses them to compute the maximum likelihood parameters of each neighbour $H'$. SEM chooses the neighbour with the best improvement in score as the new best hypothesis $H^{k+1}$, if

it improves upon $H_k$, and iterates. SEM variants of SCOOBY are an interesting future line of research.

One major drawback of Bayesian logic programs, inherited from Bayesian networks, is their acyclicity assumption. Consider a system for medical diagnosis. Two symptoms of the disease cardiac tamponade are dyspnea (breathlessness) and rapid breathing (over 20 inhalations per minute). Clearly, the two findings are related: if a patient has cardiac tamponade and has difficulty breathing, it is likely that the same patient is breathing rapidly trying to compensate for the air shortage. To cope with random variables that are related only in an undirected manner, i.e., there is no sensible cause-effect relationship, we have to resort to a trick: we add an extra constraint atom, which is always instantiated [17], see Jensen [2001]. Although, relational abstraction allows to model groups of such constraint variables, using an undirected Markov network could be more elegant as they can represent undirected probabilistic relations with non-degenerate conditional probabilities. Indeed, there exist relational and logical variants of Markov networks, namely Taskar et al.'s relational Markov networks and Domingos and Richardson's Markov logic networks. However, no research on hybrid languages combining the advantages of directed and undirected models has been conducted yet. Within the Uncertainty in AI community, models with both directed and undirected arcs are called chain graphs. So, research on 'chain logic graphs' seems to be an interesting future line of research.

Finally, Bayesian logic programs view ground atoms as random variables. In doing so, observations become a meta concept of the language: extensional Bayesian facts do not encode the state of random variables (the blood type of ann is `a`) but rather that random variables are *relevant*. In contrast, stochastic logic programs for instance, treat ground atoms as states of random variables. Thus, observations are part of the language but at the expense of losing the clear notation of random variable. Indeed, this is not a hard dichotomy; a (finite) random variable `x` with a finite domain $S(\mathtt{x}) = \{d_1, d_2, \ldots, d_n\}$ can be represented as a Markov chain $\mathtt{x}_{\mathtt{d}_1} \rightarrow \mathtt{x}_{\mathtt{d}_2} \rightarrow \ldots \rightarrow \mathtt{x}_{\mathtt{d}_n}$ encoding the mutually exclusiveness constraints among the states $d_i$. Unfortunately, this encoding is somewhat complicated and error-prone, and continuous random variables cannot be represented. Therefore, combining both views is an interesting line for future research. Avi Pfeffer's IBAL provides a first step into this direction [Pfeffer, 2001].

# Conclusions

We have described Bayesian logic programs, their representation language, their semantics, and a query-answering process, and investigated the learning of Bayesian logic programs from data.

[17] Let $A$ and $B$ two random variables influencing each other quantified by $R(A, B)$. Introduce a new boolean variable $C$ as a child of $A$ and $B$. Now, let $P(C = true|A, B) = R(A, B)$ and $P(C = no|A, B) = 1 - R(A, B)$ and enter the evidence $C = yes$.

Bayesian logic programs combine Bayesian networks with definite clause logic. The main idea of Bayesian logic programs is to establish a one-to-one mapping between ground atoms in the least Herbrand model and random variables. The least Herbrand model of a Bayesian logic program together with its direct influence relation is viewed as a (possibly infinite) Bayesian network. Bayesian logic programs inherit the advantages of both Bayesian networks and definite clause logic, including the strict separation of qualitative and quantitative aspects. Moreover, the strict separation facilitated the introduction of a graphical representation, which stays close to the graphical representation of Bayesian networks.

Indeed, Bayesian logic programs can naturally model any type of Bayesian network (including those involving continuous variables) as well as any type of "pure" Prolog program (including those involving functors). We also demonstrated that Bayesian logic programs can model hidden Markov models and stochastic grammars, and investigated their relationship to other first order extensions of Bayesian networks.

The framework for learning Bayesian logic programs is an instance of the *probabilistic learning from interpretations* setting as described in Section 2.4.3. It is unifying as it combines traditional Bayesian network learning and ILP principles. Therefore, our framework builds upon many of the results for Bayesian network learning from the Uncertainty in AI community, see e.g. [Heckerman, 1995], and upon many of the results from the ILP community. Most notably, we have adapted the EM and gradient ascent algorithms for parameter estimation, and the general structure learning mechanisms from the field of Bayesian networks, and the clausal discovery and learning from interpretations settings from ILP for probabilistic learning from interpretations.

# Related Work

Bayesian logic programs belong to the statistical relational learning line of research that extends Bayesian networks. They are motivated and inspired by the formalisms discussed in [Poole, 1993, Haddawy, 1994, Ngo and Haddawy, 1997, Jäger, 1997, Friedman et al., 1999, Koller, 1999]. We will now investigate these relationships in more detail. First, we will discuss related representational frameworks, then we will discuss related learning approaches.

### Representation

Here, we will relate the representation language and the semantics of Bayesian logic programs to other SRL approaches.

**Probabilistic logic programs** [Ngo and Haddawy, 1995, 1997] also adapt a logic program syntax, the concept of the least Herbrand model to specify the relevant random variables, and SLD resolution to develop a query-answering procedure. Whereas Bayesian logic programs view atoms as random variables, probabilistic-logic programs view them as states of random variables. For instance,

$$P(\texttt{burglary}(\texttt{Person}, \texttt{yes}) \mid \texttt{neighbourhood}(\texttt{Person}, \texttt{average})) = 0.4$$

states that the aposteriori probability of a burglary in `Person`'s house given that `Person` has an average neighbourhood is 0.4. Thus, instead of conditional probability distributions, conditional probability values are associated with clauses.

Treating atoms as states of random variables has several consequences: (1) Exclusivity constraints such as

$$\texttt{false} \leftarrow \texttt{neighbourhood}(\texttt{X}, \texttt{average}), \texttt{neighbourhood}(\texttt{X}, \texttt{bad})$$

have to be specified in order to guarantee that random variables are always in exactly one state. (2) The inference procedure is exponentially slower in time for building the support network than that for Bayesian logic programs because there is a proof for each configuration of random variable. (3) It is more difficult — if not impossible — to represent continuous random variables. (4) Qualitative, i.e., the logical component, and quantitative information, i.e., the probability values, are mixed. It is the separation of these components that enables a graphical representation for Bayesian logic programs.

Probabilistic and Bayesian logic programs are also related to Poole's framework of **probabilistic Horn abduction** [1993], which later developed into the **Independent Choice Logic** [Poole, 1997]. Probabilistic Horn abduction is "*a pragmatically-motivated simple logic formulation that includes definite clauses and probabilities over hypotheses*" [Poole, 1993]. It essentially associates distributions with certain atoms while clauses are kept purely logical. Poole's framework provides a link to abduction and assumption-based reasoning. However, as Ngo and Haddawy point out, probabilistic and therefore also Bayesian logic programs do not have as many constraints on the representation language, represent probabilistic conditional dependencies directly using clauses rather than indirectly using atoms, have a richer representational power, and their independence assumption reflects the causality of the domain.

Pfeffer [2000] introduced **probabilistic relational models**, which are based on the well-known entity/relationship model. In probabilistic relational models, the random variables are the attributes. The relations between entities are deterministic, i.e. they are only true or false. Probabilistic relational models can be described as Bayesian logic programs.

Indeed, each attribute $a$ of an entity type $E$ is a Bayesian predicate $\texttt{a}(\texttt{E})$ and each $n$-ary relation $r$ is an $n$-ary logical Bayesian predicate $\texttt{r/n}$. Probabilistic relational models consist of a qualitative dependency structure over the attributes and their associated quantitative parameters (the conditional probability densities). Getoor et al. [2002] distinguish between two types of parents of an attribute. First, an attribute $\texttt{a}(\texttt{X})$ can depend on another attribute $\texttt{b}(\texttt{X})$, e.g. the professor's popularity depends on the professor's teaching ability in the *university* domain. This is equivalent to the Bayesian clause $\texttt{a}(\texttt{X}) \mid \texttt{b}(\texttt{X})$. Second, an attribute $\texttt{a}(\texttt{X})$ possibly depends on an attribute $\texttt{b}(\texttt{Y})$ of an entity `Y` related to `X`, e.g. a student's grade in a course depends on the difficulty of the course. The relation between `X` and `Y` is described by a slot or logical relation $\texttt{s}(\texttt{X}, \texttt{Y})$. Given these logical relations, the original dependency is represented by $\texttt{a}(\texttt{X}) \mid \texttt{s}(\texttt{X}, \texttt{Y}), \texttt{b}(\texttt{Y})$. To deal with multiple ground instantiations of a single clause (with the same head ground atom), probabilistic relational models employ aggregate functions rather than combining rules as discussed earlier. Probabilistic relational models

have been extended to handle dynamic environments, i.e., distributions changing over time [Sanghai et al., 2003].

Clearly, probabilistic relational models [18] employ a more restricted logical component than Bayesian logic programs do. It is a version of the commonly used entity/relationship model, which can be represented using a (range-restricted) negation-free Datalog program [19]. Thus, they cannot employ the full power of Prolog. For instance, recall the blood type example exploiting founder information, cf. Figure 3.9. Probabilistic relational models cannot intensionally define `founder`/1 because this involves negation. Furthermore, hidden Markov models and grammars, cf. Figure 3.10 cannot be (at least directly) represented because this involves functors. As another example, consider the case where the classification of a molecule depends on whether there is a path between two atoms or not. The definition of `path`/2 cannot be represented intensionally within a probabilistic relational model because this requires recursion. In contrast, Bayesian logic programs have the full expressivity of definite clause logic and, therefore, of a universal Turing machine. Indeed, general definite clause logic (using functors) is undecidable. The functor-free fragment of definite clause logic, however, is decidable.

Jäger [1997] introduced **relational Bayesian networks**, which are Bayesian networks where the nodes are predicate symbols. The states of these random variables are possible interpretations of the symbols over an arbitrary, finite domain (here we only consider Herbrand domains), i.e. the random variables are set-valued. The inference problem addressed by Jäger asks for the probability that an interpretation contains a ground atom. Thus, relational Bayesian networks are viewed as Bayesian networks where the nodes are the ground atoms and have the domain $\{true, false\}$[20]. The key difference between relational Bayesian networks and Bayesian logic programs is that the quantitative information is specified by so called probability formulas. These formulas employ the notion of combination functions, functions that map every finite multiset with elements from $[0, 1]$ into $[0, 1]$, as well as that of equality constraints[21]. Let $F(cancer)(x)$ be $noisy\_or\{comb_\Gamma\{exposed(x, y, z) \mid z; true\} \mid y; true\}$ . This formula states that that for any specific organ $y$, multiple exposures to radiation have a cumulative but independent effect on the risk of developing cancer of $y$. Thus, a probability formula not only specifies the distribution but also the dependency structure. As a consequence and also because of the computational power of combining rules, a probability formula is easily expressed as a set of Bayesian clauses: the head of the Bayesian clauses is the corresponding Bayesian atom and the bodies consist of all maximally generalized Bayesian atoms occurring in the probability formula. Now the combining rule can select the right ground atoms and simulate the probability formula. This is always possible because the Herbrand base is finite. E.g. the clause `cancer(X) | exposed(X,Y,Z)` together with the right combining rule and associated conditional probability distribution models the example formula.

[18] Several extensions to treat *existential uncertainty*, *referential uncertainty*, and *domain uncertainty* exist.

[19] Datalog programs are definite clause programs without function symbols.

[20] It is possible, but complicated to model domains having more than two values.

[21] To simplify the discussion, we will further ignore these equality constraints here.

Recently, Milch et al. [2004, 2005] extended Bayesian logic programs and probabilistic relational models to **Bayesian logic** (BLOG). Every BLOG model specifies a unique probability distribution over first-order model structures that can contain varying and unbounded numbers of objects. The key idea is a generative process that constructs worlds by adding objects whose existence and properties depend on those of objects already created. Objects can either be guaranteed, meaning the extension is fixed, or they can be generated from a distribution. Thus, BLOG overcomes the closed world and unique name assumptions employed within Bayesian logic programs.

Carbonetto et al. [2005] introduced a nonparametric extension of BLOG. BLOG specifies a prior over the number of objects. In many domains, however, it is unreasonable for the user to suggest such a proper, data-independent prior. In contrast, in **NP-BLOG (nonparametric BLOG)**, Carbonetto et al. use Dirichlet processes to cast distributions over unknown objects and their attributes. This, in turn, resolves some difficulties in model selection and inference caused by varying numbers of objects because it handles distributions over unbounded sets of objects.

Laskey and Costa [2005] introduced **multi-entity Bayesian networks** (MEBN). MEBNs represent the world as comprised of entities that have attributes and are related to other entities. Whereas Bayesian logic programs represent and quantify the relations among random variables using Horn clauses, MEBNs uses so-called MFrags. MFrags contain nodes arranged in a directed graph. Nodes represent random variables and arcs represent direct dependency relationships; and local distributions specify conditional probability distributions. Each node has an associated random variable label and a parameterized list of arguments. Entity identifiers are substituted for arguments to form instances of the random variables.

In addition to extensions of Bayesian networks, several other probabilistic models have been extended to the first-order or relational case: Sato [1995] introduced **distributional semantics** in which ground atoms are seen as random variables over $\{true, false\}$. Probability distributions are defined over the ground facts of a program and propagated over the Herbrand base of the program using the clauses. Consider the following PRISM program (inspired by Sato and Y.Kameya [2004]) specifying a distribution over ground atoms of the form `toss(l,n)` such that `l` is a list of outcomes of `n` coin tosses:

```
values(coin,[a,b]).   % discrete random variable named coin
                      % with domain {a,b}
:- set_sw(coin,[0.6+0.4])   % distribution over {a,b}

toss(N,[A|C]) :- N>0, msw(coin,A), msw(coin,B),A==B,
                 N1 is N-1,test(N1,C).
toss(0,[]).
```

Here, `msw(coin,.)` implements a single stochastic coin toss resulting with probability 0.6 in `a` and with probability 0.4 in `b`. In principle, the probability of a derivation is the product of all `msw` results used in the derivation. PRISM programs, however, can fail, which in turn affects the distribution. Consider querying `:- toss(2,X)`. The clause head `toss(N,[A|C])` unifies, and the goal `msw` is executed two time, i.e., values for `A` and `B` are chosen randomly according to the distribution specified. If `a` respectively

`b` are sampled, the goal `A==B` fails, so does `:- toss(2,X).` In turn, no observation is generated. Therefore, the probability of a single proof is its derivation probability normalized by the sum of all its alternative proofs. This is akin to the semantics of **stochastic logic programs** [Muggleton, 1996, Cussens, 1999], which we have discussed in Section 2.3.2. As a reminder, stochastic logic programs lift probabilistic context-free grammars to the first order case by replacing production rules with probability values with clauses labeled with probability values.

Taskar et al.'s **relational Markov networks** (RMN) [2002] extend Markov networks, see e.g. [Pearl, 1991], by providing a relational language for describing clique structures and enforcing parameter sharing at the template level. A key concept of RMNs are relational clique templates, which specify the structure of a Markov network. A clique template $C$ is similar to a SQL database query. It selects nodes from a Markov network and connects them into cliques. Each clique template $C$ is additionally associated with a potential function $\phi_C(\mathbf{v}_c)$ that maps values $\mathbf{v}_C$ of random variables to nonnegative real numbers. To perform inference and estimate parameters, an RMN is unrolled into a Markov network; the connections among the nodes are built by applying the clique templates to the data; each template $C$ can result in several cliques that have identical structure and share the same potential function $\phi_C(\cdot)$. The resulting Markov network factorizes the distribution as

$$P(\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C \in \mathcal{C}} \prod_{\mathbf{v}_C \in C} \phi_C(\mathbf{v}_C)$$

where $Z(\mathbf{x})$ is the normalizing partition function. Usually, the clique potentials are represented as a log-linear combination of features

$$\phi_C(\mathbf{v}_c) = e^{\mathbf{w}_C^t \cdot \mathbf{f}_C(\mathbf{v}_c)}$$

where $\mathbf{f}_C(\cdot)$ defines a feature vector for clique $C$ and $\mathbf{w}_C^t$ is the transposed corresponding weight vector. RMNs can be viewed as the undirected version of PRMs. Whereas inference in Bayesian networks is NP hard [Cooper, 1990], inference in Markov network is #P-complete [22] [Roth, 1998]. Therefore, one usually resorts to approximate inference techniques such as Gibbs sampling [Geman and Geman, 1984], where each variable is sampled in turn given its direct neighbours, and belief propagation [Yedidia et al., 2001]. In contrast to Bayesian networks, maximum aposteriori and maximum likelihood estimates of Markov networks cannot be represented in closed form in the fully observable case. However, as the log-likelihood is a concave function of the weights, they can be efficiently be computed using standard gradient=based approaches.

Recently, Domingos and Richardson [2004] introduced **Markov logic networks** (MLNs), see Section 2.3.1 for a discussion, which upgrade Markov networks to the

[22] #P is the class of functions $f : \{0,1\}^n \mapsto N$ for which there is a nondeterministic polynomial time Turing machine $M$ such that for every word $w$, $M(w)$ has exactly $f(w)$ accepting paths. It is natural to expect that for NP complete problems the corresponding counting problem is in #P but it is also possible to get #P problems out of problems, which are polynomial solvable.

first order case. Therefore, one can view MLNs as undirected version of Bayesian logic programs. Similar to RMNs, an MLN is unrolled into a Markov network. Jäger [2005] recently showed that MLNs can be essentially represented as RBNs and, in turn, as Bayesian logic programs.

Neville and Jensen [2004] proposed **relational dependency networks** (RDNs), which are relational upgrades of dependency networks [Heckerman et al., 2000] (DNs). DNs approximate the joint distribution with a set of conditional probability distributions, which are learned independently. More precisely, a DN is essentially a fully connected, directed graph over a set of nodes $\mathbf{X}$. With each node $X$, a conditional probability distribution $\mathbf{P}(X|\mathbf{X}\setminus\{X\})$ is associated. In general, a DN may not always represent a joint distribution. Therefore, Heckerman et al. propose a Gibbs sampling approach for approximative inference. Neville and Jensen now replace the conditional probability distribution by relational probability trees [Neville et al., 2003] for each predicate.

Santos Costa et al. [2003a] introduced **CLP(BN)**, which is a constraint logic programming (CLP) approach to define joint probability distributions over missing values in a logic program. In general, logic variables are said to be constrained if they are bound to one or more constraints. Here, constraints are restrictions on the set of constants logical variables can take. CLP(BN) replaces hard constraints, which are typically employed within CLP, by Bayesian ones. Consider the following example within the *university* domain:

```
grade(Reg,Grade) :-  reg(Reg,Course,Student),
                     difficulty(Course,Dif),
                     intelligence(Student,Int),
                     {Grade = grade(Reg) with p(
                     [a,b,c],[0.4,0.0,0.4,0.0,
                              0.4,0.1,0.4,0.1,
                              0.2,0.9,0.2,0.9],[Dif,Int])}.
```

This CLP(BN) rule says that a registration's (`Ref`) grade `Grade` probabilistically depends on the course's difficulty `Dif` and on the student's intelligence `Int`, which both can take only two values. The predicate `p`/3 encodes the conditional probability distribution $\mathbf{P}(\texttt{Grade}|\texttt{Dif},\texttt{Int})$ where the domain of `Grade` is $[\texttt{a},\texttt{b},\texttt{c}]$. For a query, say `:- grade(r2,Grade)`, a Bayesian network is constructed where `grade`(`r2`) depends on `diff`(`course`) and `int`(`student`). In other words, logical variables are treated as random variables among which a Bayesian network is constructed.

Most probabilistic ILP representation languages adapt Horn clauses. Vennekens et al. [2004] recently introduced **logic programs with annotated disjunctions** (LPADs), which are disjunctive logic programs where disjunctive heads encode probability distributions conditioned on the bodies. More precisely, a LPAD consists of a set of rules of the form

$$(\mathtt{h}_1,\alpha_1)\vee\ldots\vee(\mathtt{h}_n,\alpha_n)\leftarrow\mathtt{b}_1,\ldots,\mathtt{b}_m$$

where the $\mathtt{h}_i$ and $\mathtt{b}_i$ are atoms respectively literals, and the $\alpha_i$ are probability values summing to 1. The meaning of such a rule is as follows: *if the body is true, then atom* $\mathtt{h}_i$

*(and only* $\mathtt{h_i}$*) is true with probability* $\alpha_i$. The semantics is given by the grounding of a LPAD. Each ground clause corresponds to probabilistic choice among the ground head atoms. By choosing one possibility for each ground clause, one gets a definite clause program. Assuming independency between the choices, one can assign a probability to each such ground program, i.e., we multiply all probability values.

Recently, Shen and Yang [2005] adapted the view on ground atoms as random variables taken in Bayesian logic programs to induce logically structured dynamics Bayesian networks.

Finally, Bayesian logic programs are related — to some extent — to the **BUGS** language [Gilks et al., 1994], which aims at carrying out hierarchical Bayesian modeling and Bayesian inference using Gibbs sampling. It uses concepts of imperative programming languages such as for-loops to model regularities in probabilistic models. Therefore, the relation between Bayesian logic programs and BUGS is akin to the general relation between declarative and imperative languages. This holds in particular for relational domains such as those used in this chapter. Without the notion of objects and relations among objects family trees are hard to represent: BUGS uses traditional indexing to group together random variables (e.g. $X_1, X_2, \ldots$ all having the same distribution), whereas Bayesian logic programs use definite clause logic.

### Parameter Estimation

In the past few years, a number of techniques for *learning* logical and relational extensions of probabilistic representations have been developed. Most of these techniques (with a few notable exceptions for probabilistic relation models, object-oriented Bayesian networks, stochastic logic programs, and Markov logic networks, see below) address the parameter estimation problem only. Furthermore, most of these approaches adapt the EM-algorithm for estimating the parameters: Koller and Pfeffer [1997] did this for Ngo and Haddawy's probabilistic logic programs, Cussens [2001] for stochastic logic programs, Sato and Kameya [2001] for PRISM, Gilks et al. [1994] for BUGS, and Langseth and Bangsø [2001] for object-oriented Bayesian networks. Our work differs in two respects. First, we do not only adapt EM but also show how gradient approaches can be used. As argued in Section 4.3.2, gradient ascent methods are — in certain situations — preferable. Second, as argued above, the underlying knowledge representation frameworks are quite different, certainly for BUGS, object-oriented Bayesian networks, PRISM and stochastic logic programs. Therefore, the closest work is certainly that by Koller and Pfeffer [1997]. Further parameter estimation methods have been developed by Taskar et al. [2001] and Flach and Lachiche [1999] for applications in clustering. Taskar et al. employ probabilistic relational models and Flach and Lachiche essentially apply the Näive Bayes algorithm. Taskar et al. [2002] considered the task of discriminative learning of relational Markov models. More precisely, they propose a gradient-based optimization method for maximizing the conditional log-likelihood of some target variables on some observation. Later on, Taskar et al. [2004b,a, 2005] introduced max-margin formulations and efficient solutions based on quadratic programming techniques for the same problem.

**Structure Learning**

Structure learning for probabilistic relational and logical representations has been addressed by Getoor [2001] for probabilistic relational models, by Muggleton [2002] (and references in there) for stochastic logic programs, by Neville and Jensen [2004] for relational dependency networks, by Riguzzi [2004] for logic programs with annotated disjunctions, and by Kok and Domingos [2005] for Markov logic networks.

Getoor [2001] showed how to learn the structure of probabilistic relational models [Pfeffer, 2000], which are based on the entity-relationship model, by adapting the Structural-EM algorithm for learning Bayesian network. The Structural-EM [Friedman, 1998] algorithm is a variant of the standard EM algorithm for maximum likelihood parameter estimation for learning the structure. The key idea is that the expected counts are not computed anew for every structure that is proposed, but only after several iterations. This leads to an improved efficiency. It should be clear that the structural EM algorithm could straightforwardly be incorporated in our framework. The most important differences between Getoor's approach and ours are due to the differences between the underlying entity-relationship model and logic programs (cf. also above). To refine the structure of the entity-relationship model, Getoor employs refinements that add and delete deterministic slot-chains (a kind of lookaheads) to establish the *influenced by* relation, and that have been specifically designed for use with probabilistic relational models. Whereas Getoor approaches the problem from a Bayesian network learning side, we employ traditional ILP principles. The same holds for structural learning of object-oriented Bayesian networks [Bangsø et al., 2001].

There is some work on learning the structure of stochastic logic programs [Muggleton, 2002]. The differences between this work and ours are are those between *probabilistic learning from entailment* and *probabilistic learning from interpretations*. The former setting, as argued in Section 2.5, is expected to be harder and this also explains why structure learning for stochastic logic programs has so far been restricted to learning missing clauses for a single predicate. More powerful settings for learning the structure of stochastic logic programs would amount — at the logical level — to perform theory revision, a known hard problem in inductive logic programming [De Raedt et al., 1993, Wrobel, 1996]. Combining this with probabilities seems even harder. For this reason, we proposed to learn stochastic logic programs from proof-banks, see Section 2.4.4.

Riguzzi [2004] proposed to learning logic programs with annotated disjunctions (LPADS) from a sets of interpretations annotated with their probabilities. The learning algorithm first finds all the disjunctive clauses that are true in all interpretations, then a probability value is associated with each disjunct in the heads, and finally it is decided how to combine the clauses to form a LPAD by solving a constraint satisfaction problem. Thus, this learning approach involves multiple separated steps and, hence, is not an integrated approach.

The learning algorithm for relational dependency networks (RDNs) is elegant and simple. Neville and Jensen [2004] just used a learning algorithm for relational probability trees [Neville et al., 2003] to independently learn a set of conditional relational models.

For Markov logic networks, Kok and Domingos [2005] developed a structure learning algorithm within the probabilistic learning from interpretation setting. They basically employ the traditional ILP refinement operators to traverse the hypothesis space. Starting from a set of unit clauses, they iteratively add to or delete literals from clauses, flip the sign of literals. To overcome the heavy computational demands of parameter estimation of Markov network, Kok and Domingos employed a weighted version of the pseudo-log-likelihood. For more details we refer to Section 2.4.3. Recently, variants have been developed for discriminatively learning MLNs [Singla and Domingos, 2005] and for learning MLNs of relational state models evolving over time [Sanghai et al., 2005].

Revoredo and Zaverucha [2002] and Paes et al. [2005] investigated theory revision for Bayesian logic programs.

Part II

# Probabilistic ILP over Time

Many real world applications such as language modeling in speech recognition, music modeling, machine translation, and sequence analysis in bioinformatics, require to model probability distributions over sets of strings, sequences, words, phrases, and trees. Bayesian logic programs as presented in Part I, however, are not customized for modeling the evolution of the state of the environment over time. Indeed, discrete *time* can be considered as yet another Bayesian predicate, e.g., `state(next(Time))|state(Time)`. This view, however, often does not heal the curse of dimensionality, especially when states themselves are structured: the set of possible state trajectories grows exponentially over time.

To this aim, Part II introduces *logical hidden Markov models* in Chapter 5. Logical hidden Markov models can be viewed as a special-purpose probabilistic ILP approach that reasons efficiently with sequential data. Traditional probabilistic models of sequences such as *hidden Markov models* consider sequences of flat symbols only. Many real world sequences such as protein secondary structures and shell logs, however, exhibit a rich internal structure. Logical hidden Markov models deal with sequences of structured symbols in the form of logical atoms. Solutions to the central inference problems — evaluation, most likely hidden state sequence and parameter estimation — are presented in Chapter 6. Part II concludes with presenting SAGEM in Chapter 7, which is a method for selecting *logical hidden Markov models* from data that combines generalized EM, which optimizes parameters, with ILP techniques for structure search.

This page intentionally left blank

§ 5

# Logical Hidden Markov Models *

*... in which logical hidden Markov models are introduced, their semantics are defined, and the design choices underlying logical hidden Markov models are discussed ...*

Hidden Markov models [Rabiner and Juang, 1986] (HMMs) are extremely popular for analyzing sequential data. Application areas include computational biology, user modeling, speech recognition, empirical natural language processing, and robotics. Despite their successes, HMMs have a major weakness: they handle only sequences of flat, i.e., unstructured symbols. Yet, in many applications the symbols occurring in sequences are structured.

**Example Domain II** (UNIX Command Sequences) *Sequences of* UNIX *commands tell a lot about the user herself since users tend to respond in a similar manner to similar situations, leading to repeated sequences of actions. For instance, LaTeX users frequently run* EMACS *to edit their LaTeX files and afterwards compile the edited file using LaTeX. The existence of command alias mechanisms in many* UNIX *command interpreters also supports the idea that users tend to enter many repeated sequences of commands. Thus,* UNIX *command sequences carry a lot information, which can be used to automatically construct user profiles. These user profiles can subsequently be employed to predict the next command in the sequence [Davison and Hirsh, 1998], to suggest personalized command aliases [Jacobs and Blockeel, 2001], to classify a command sequence into a user category [Korvemaker and Greiner, 2000, Jacobs and Blockeel, 2001], and to detect anomalous behavior [Lane, 1999].* ○

**Example 5.1** Consider the UNIX command sequence `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, .... ○

Indeed, UNIX commands are essentially structured as they are composed of their names such as `emacs`, `ls`, `xdvi`, `latex` and their arguments such as `lohmms.tex`. Traditional HMMs cannot easily deal with this type of structured sequences. Indeed, applying HMMs requires either

- ignoring the structure of the commands (i.e., the arguments), or
- taking all possible parameters explicitly into account.

The former approach results in a serious information loss; the latter leads to a combinatorial explosion in the number of symbols and parameters of the HMM and as a consequence inhibits generalization.

The above sketched problem with HMMs is akin to the problem of dealing with structured examples in traditional machine learning algorithms as studied in the fields of inductive logic programming [Muggleton and De Raedt, 1994] and multi-relational

* Builds on [Kersting et al., 2002, Raiko et al., 2002, Kersting et al., 2003b, 2006].

learning [Džeroski and Lavrač, 2001]. Here, we propose an (inductive) logic programming framework, logical hidden Markov models (LOHMMs), that upgrades HMMs to deal with structure. The key idea underlying logical hidden Markov models is to employ logical atoms as structured (output and state) symbols.

**Example 5.2** Using logical atoms, the above UNIX command sequence can be represented as $\texttt{emacs}(\texttt{lohmms},\texttt{tex}), \texttt{ls}, \texttt{latex}(\texttt{lohmms},\texttt{tex}), \ldots$ ○

There are two important motivations for using logical atoms at the symbol level. First, *variables* in the atoms allow one to make abstraction of specific symbols. Second, *unification* allows one to share information among states.

**Example 5.3** In $\texttt{emacs}(\texttt{X.tex}), \texttt{latex}(\texttt{X.tex})$ the logical atom $\texttt{emacs}(\texttt{X},\texttt{tex})$ represents all files X that a LaTeX user tex could edit using emacs. Furthermore, it denotes that the same file is used as an argument for both Emacs and LaTeX. ○

## 5.1 Representation Language

The logical component of a traditional HMM corresponds to a *Mealy machine* [Hopcroft and Ullman, 1979], i.e., a finite state machine where the output symbols are associated with transitions. This is essentially a propositional representation because the symbols used to represent states and output symbols are propositional/flat, i.e. not structured. Many real-world domains such as UNIX command sequences, however, exhibit a rich internal structure.

The key idea underlying logical hidden Markov models is to replace these flat symbols by abstract symbols. Consider the UNIX command shell domain. An abstract symbol A is — by definition — a logical atom. It is abstract in that it represents the set of all ground, i.e., variable-free atoms of A over the alphabet $\Sigma$, denoted by $G_\Sigma(\texttt{A})$. Ground atoms then play the role of the traditional symbols used in a HMMs.

**Example 5.4** Consider the alphabet $\Sigma_1$, which has as constant symbols tex, dvi, hmm1, and lohmm1 and as relation symbols emacs/2, ls/1, xdvi/1, latex/2. Then the atom $\texttt{emacs}(\texttt{File},\texttt{tex})$ represents the set $\{\texttt{emacs}(\texttt{hmm1},\texttt{tex}), \texttt{emacs}(\texttt{lohmm1},\texttt{tex})\}$. We assume that the alphabet is typed to avoid useless instantiations such as $\texttt{emacs}(\texttt{tex},\texttt{tex})$). ○

The use of atoms instead of flat symbols allows us to analyze logical and structured sequences such as $\texttt{emacs}(\texttt{hmm1},\texttt{tex}), \texttt{latex}(\texttt{hmm1},\texttt{tex}), \texttt{xdvi}(\texttt{hmm1},\texttt{dvi})$.

**Definition 5.5** *Abstract transitions* are expressions of the form

$$p : \texttt{H} \xleftarrow{\texttt{O}} \texttt{B}$$

where $p \in [0,1]$, and H, B and O are atoms. All variables are implicitly assumed to be universally quantified, i.e., the scope of variables is a single abstract transition. ○

The atoms H and B represent abstract states and O represents an abstract output symbol. The semantics of an abstract transition $p : \texttt{H} \xleftarrow{\texttt{O}} \texttt{B}$ is that if one is in one of the states in $G_\Sigma(\texttt{B})$, say $\texttt{B}\theta_\texttt{B}$, one will go with probability $p$ to one of the states in $G_\Sigma(\texttt{H}\theta_\texttt{B})$, say $\texttt{H}\theta_\texttt{B}\theta_\texttt{H}$, while emitting a symbol in $G_\Sigma(\texttt{O}\theta_\texttt{B}\theta_\texttt{H})$, say $\texttt{O}\theta_\texttt{B}\theta_\texttt{H}\theta_\texttt{O}$.

**Example 5.6** Consider $c \equiv 0.8 : \texttt{xdvi}(\texttt{File}, \texttt{dvi}) \xleftarrow{\texttt{latex(File)}} \texttt{latex}(\texttt{File}, \texttt{tex})$. In general H, B and O do not have to share the same predicate. This is only due to the nature of our running example. Assume now that we are in state $\texttt{latex}(\texttt{hmm1}, \texttt{tex})$, i.e. $\theta_{\texttt{B}} = \{\texttt{File}/\texttt{hmm1}\}$. Then $c$ specifies that there is a probability of 0.8 that the next state will be in $G_{\Sigma_1}(\texttt{xdvi}(\texttt{hmm1}, \texttt{dvi})) = \{\texttt{xdvi}(\texttt{hmm1}, \texttt{dvi})\}$ (i.e., the probability is 0.8 that the next state will be $\texttt{xdvi}(\texttt{hmm1}, \texttt{dvi})$), and that one of the symbols in $G_{\Sigma_1}(\texttt{latex}(\texttt{hmm1})) = \{\texttt{latex}(\texttt{hmm1})\}$ ( i.e., $\texttt{latex}(\texttt{hmm1})$) will be emitted. Abstract states might also be more complex such as $\texttt{latex}(\texttt{file}(\texttt{FileStem}, \texttt{FileExtension}), \texttt{User})$ ○

The above example was simple because $\theta_{\texttt{H}}$ and $\theta_{\texttt{O}}$ were both empty. The situation becomes more complicated when these substitutions are not empty. Then, the resulting state and output symbol sets are not necessarily singletons.

**Example 5.7** Indeed, for $0.8 : \texttt{emacs}(\texttt{File}', \texttt{dvi}) \xleftarrow{\texttt{latex(File)}} \texttt{latex}(\texttt{File}, \texttt{tex})$ the resulting state set would be the set of subsumed ground states of $\texttt{emacs}(\texttt{File}', \texttt{dvi})$, i.e., $G_{\Sigma_1}(\texttt{emacs}(\texttt{File}', \texttt{dvi})) = \{\texttt{emacs}(\texttt{hmm1}, \texttt{tex}), \texttt{emacs}(\texttt{lohmm1}, \texttt{tex})\}$. ○

Thus, the transition is non-deterministic because there are two possible resulting states. We therefore need a mechanism to assign probabilities to these possible alternatives.

**Definition 5.8** The selection distribution $\mu$ specifies for each abstract state and observation symbol A over the alphabet $\Sigma$ a distribution $\mu(\cdot \mid \texttt{A})$ over $G_\Sigma(\texttt{A})$. ○

**Example 5.9** To continue our example, consider the selection probability $\mu(\texttt{emacs}(\texttt{hmm1}, \texttt{tex}) \mid \texttt{emacs}(\texttt{File}', \texttt{tex})) = 0.4$ and the selection probability $\mu(\texttt{emacs}(\texttt{lohmm1}, \texttt{tex}) \mid \texttt{emacs}(\texttt{File}', \texttt{tex})) = 0.6$. Then there would be a probability of $0.4 \times 0.8 = 0.32$ that the next state is $\texttt{emacs}(\texttt{hmm1}, \texttt{tex})$ and of 0.48 that it is $\texttt{emacs}(\texttt{lohmm1}, \texttt{tex})$. ○

Taking $\mu$ into account, the meaning of an abstract transition $p : \texttt{H} \xleftarrow{\texttt{O}} \texttt{B}$ can be summarized as follows:

**Semantics 5.10** (Abstract Transition) Let $\texttt{B}\theta_{\texttt{B}} \in G_\Sigma(\texttt{B})$, $\texttt{H}\theta_{\texttt{B}}\theta_{\texttt{H}} \in G_\Sigma(\texttt{H}\theta_{\texttt{B}})$ and $\texttt{O}\theta_{\texttt{B}}\theta_{\texttt{H}}\theta_{\texttt{O}} \in G_\Sigma(\texttt{O}\theta_{\texttt{B}}\theta_{\texttt{H}})$. Then the model makes a transition from state $\texttt{B}\theta_{\texttt{B}}$ to $\texttt{H}\theta_{\texttt{B}}\theta_{\texttt{H}}$ and emits symbol $\texttt{O}\theta_{\texttt{B}}\theta_{\texttt{H}}\theta_{\texttt{O}}$ with probability

$$p \cdot \mu(\texttt{H}\theta_{\texttt{B}}\theta_{\texttt{H}} \mid \texttt{H}\theta_{\texttt{B}}) \cdot \mu(\texttt{O}\theta_{\texttt{B}}\theta_{\texttt{H}}\theta_{\texttt{O}} \mid \texttt{O}\theta_{\texttt{B}}\theta_{\texttt{H}}). \tag{5.1}$$

○

To represent $\mu$, any probabilistic representation can - in principle - be used, e.g. a Bayesian network or a Markov chain. Throughout the remainder of the thesis, however, we will use a *naïve Bayes* approach. More precisely, we associate to each argument of a relation $\texttt{r}/m$ a finite domain $\text{dom}_i^{\texttt{r}/m}$ of constants and a probability distribution $P_i^{\texttt{r}/m}$ over $\text{dom}_i^{\texttt{r}/m}$. Let $\text{vars}(\texttt{A}) = \{\texttt{V}_1, \ldots, \texttt{V}_l\}$ be the variables occurring in an atom A over $\texttt{r}/m$, and let $\sigma = \{\texttt{V}_1/\texttt{s}_1, \ldots \texttt{V}_l/\texttt{s}_l\}$ be a substitution grounding A.

Each $\mathtt{V}_j$ is then considered a random variable over the domain $D^{\mathtt{r}/m}_{\arg(\mathtt{V}_j)}$ of the argument $\arg(\mathtt{V}_j)$ it appears first in. Then,

$$\mu(\mathtt{A}\sigma \mid \mathtt{A}) = \prod_{j=1}^{l} P^{\mathtt{r}/m}_{\arg(\mathtt{V}_j)}(\mathtt{s}_j) \ .$$

**Example 5.11** In our running example, $\mu(\mathtt{emacs}(\mathtt{hmm1},\mathtt{tex}) \mid \mathtt{emacs}(\mathtt{F},\mathtt{E}))$, is computed as the product of $P_1^{\mathtt{emacs}/2}(\mathtt{hmm1})$ and $P_2^{\mathtt{emacs}/2}(\mathtt{tex})$. ∘

Thus far the semantics of a single abstract transition has been defined. A logical hidden Markov model usually consists of multiple abstract transitions and this creates a further complication.

**Example 5.12** Consider $0.8 : \mathtt{latex}(\mathtt{File},\mathtt{tex}) \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs}(\mathtt{File},\mathtt{tex})$ and $0.4 : \mathtt{dvi}(\mathtt{File}) \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs}(\mathtt{File},\mathtt{User})$. These two abstract transitions make conflicting statements about the state resulting from $\mathtt{emacs}(\mathtt{hmm1},\mathtt{tex})$. Indeed, according to the first transition, the probability is 0.8 that the resulting state is $\mathtt{latex}(\mathtt{hmm1},\mathtt{tex})$ and according to the second one it assigns 0.4 to $\mathtt{xdvi}(\mathtt{hmm1})$. ∘

There are essentially two ways to deal with this situation. On the one hand, one might want to combine and normalize the two transitions and assign a probability of $\frac{2}{3}$ respectively $\frac{1}{3}$ . On the other hand, one might want to have only one rule firing. We chose the latter option because it allows us to consider transitions more independently, it simplifies learning, and it yields locally interpretable models. We employ the subsumption (or generality) relation among the B-parts of the two abstract transitions. Indeed, the B-part of the first transition $\mathtt{B}_1 = \mathtt{emacs}(\mathtt{File},\mathtt{tex})$ is more specific than that of the second transition $\mathtt{B}_2 = \mathtt{emacs}(\mathtt{File},\mathtt{User})$ because there exists a substitution $\theta = \{\mathtt{User}/\mathtt{tex}\}$ such that $\mathtt{B}_2\theta = \mathtt{B}_1$, i.e., $\mathtt{B}_2$ subsumes $\mathtt{B}_1$. Therefore $G_{\Sigma_1}(\mathtt{B}_1) \subseteq G_{\Sigma_1}(\mathtt{B}_2)$ and the first transition can be regarded as more informative than the second one. It should therefore be preferred over the second one when starting from $\mathtt{emacs}(\mathtt{hmm1},\mathtt{tex})$. We will also say that the first *transition* is *more specific* than the second one. Remark that this *generality* relation imposes a partial order on the set of all transitions. These considerations lead to the strategy of only considering the maximally specific transitions that apply to a state in order to determine the successor states. This implements a kind of exception handling or default reasoning and is akin to Katz's [1987] *back-off* $n$-gram models. In back-off $n$-gram models, the most detailed model that is deemed to provide sufficiently reliable information about the current context is used. That is, if one encounters an $n$-gram that is not sufficiently reliable, then back-off to use an $(n-1)$-gram; if that is not reliable either then back-off to level $n-2$, etc.

The conflict resolution strategy will work properly provided that the bodies of all maximally specific transitions (matching a given state) represent the same abstract state. This can be enforced by requiring the *generality* relation over the B-parts to be closed under the *greatest lower bound* (glb) for each predicate, i.e., for each pair $\mathtt{B}_1,\mathtt{B}_2$ of bodies, such that $\theta = \mathrm{mgu}(\mathtt{B}_1,\mathtt{B}_2)$ exists, there is another body B (called lower bound), which subsumes $\mathtt{B}_1\theta$ (and therefore also $\mathtt{B}_2\theta$) and is subsumed by $\mathtt{B}_1,\mathtt{B}_2$,

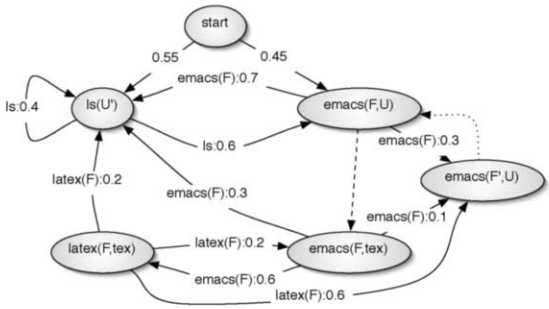



**Figure 5.1.** A logical hidden Markov model. Abstract states are represented by blue nodes. Arrows between nodes denote abstract transition. The abstract emissions and transition probabilities are associate with the arrows. Dotted arrows denote 'must-follow' links; dashed arrows the 'more-general-than' relation.

and if there is any other lower bound then it is subsumed by $\mathtt{B}$. E.g., if the body of the second abstract transition in our example is $\mathtt{emacs}(\mathtt{hmm1}, \mathtt{User})$ then the set of abstract transitions would not be closed under glb.

Finally, in order to specify a prior distribution over states, we assume a finite set $\Upsilon$ of clauses of the form $p : \mathtt{H} \leftarrow \mathtt{start}$ using a distinguished $\mathtt{start}$ symbol such that $p$ is the probability of the logical hidden Markov model to start in a state of $G_\Sigma(\mathtt{H})$.

By now we are able to formally define logical hidden Markov models.

**Definition 5.13** A logical hidden Markov model (LOHMM) is a tuple $(\Sigma, \mu, \Delta, \Upsilon)$ where $\Sigma$ is a logical alphabet, $\mu$ a selection probability over $\Sigma$, $\Delta$ is a set of abstract transitions, and $\Upsilon$ is a set of abstract transitions encoding a prior distribution. Let $\mathbf{B}$ be the set of all atoms that occur as body parts of transitions in $\Delta$. We assume $\mathbf{B}$ to be closed under glb and require

$$\forall \mathtt{B} \in \mathbf{B} : \sum\nolimits_{p:\mathtt{H} \xleftarrow{0} \mathtt{B} \in \Delta} p = 1.0 \tag{5.2}$$

and that the probabilities $p$ of clauses in $\Upsilon$ sum up to 1.0 . ∘

Logical hidden Markov models can also be represented graphically. Figure 5.1 contains an example. The underlying language $\Sigma_2$ consists of $\Sigma_1$ together with the constant symbol $\mathtt{other}$, which denotes a user that does not employ LaTeX. In this graphical notation, nodes represent abstract states and *black tipped arrows* denote abstract transitions. *White tipped arrows* are used to represent meta knowledge. More precisely, *white tipped, dashed arrows* represent the generality or subsumption ordering between abstract states. If we follow a transition to an abstract state with an outgoing *white tipped, dotted arrow* then this dotted arrow will always be followed. Dotted arrows are needed because the same abstract state can occur under different circumstances. Consider the transition $p : \mathtt{latex}(\mathtt{File}', \mathtt{User}') \xleftarrow{\mathtt{latex(File)}} \mathtt{latex}(\mathtt{File}, \mathtt{User})$. Even though the atoms in the

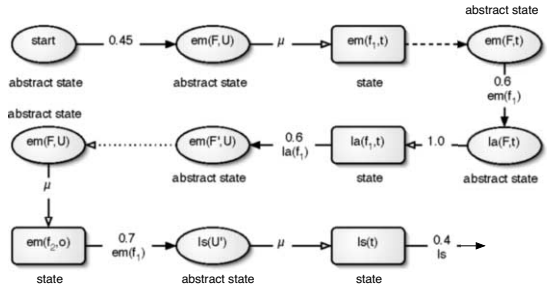



**Figure 5.2.** Generating the observation sequence `emacs(hmm1)`, `latex(hmm1)`, `emacs(lohmm1)`, `ls` by the logical hidden Markov model in Figure 5.1. The command `emacs` is abbreviated by `em`, $\mathtt{f}_1$ denotes the filename `hmm1`, $\mathtt{f}_2$ represents `lohmm1`, `t` denotes a `tex` user, and `o` some `other` user. White solid arrows indicate selections.

head and body of the transition are syntactically different they represent the same abstract state. To accurately represent the meaning of this transition we cannot use a black tipped arrow from $\mathtt{latex}(\mathtt{File},\mathtt{User})$ to itself, because this would actually represent the abstract transition $p : \mathtt{latex}(\mathtt{File},\mathtt{User}) \xleftarrow{\mathtt{latex(File)}} \mathtt{latex}(\mathtt{File},\mathtt{User})$.

## 5.2 Semantics

Hidden Markov models are a special cases of logical hidden Markov models in which $\Sigma$ contains only relation symbols of arity zero and the selection probability is irrelevant. Thus, logical hidden Markov models directly generalize hidden Markov models.

Furthermore, the graphical representation clarifies that logical hidden Markov models are generative models. Let us explain how the model in Figure 5.1 would generate the observation sequence $\mathtt{emacs}(\mathtt{hmm1})$, $\mathtt{latex}(\mathtt{hmm1})$, $\mathtt{emacs}(\mathtt{lohmm1})$, $\mathtt{ls}$ (cf. Figure 5.2). It chooses an initial abstract state, say $\mathtt{emacs}(\mathtt{F},\mathtt{U})$. Since both variables `F` and `U` are uninstantiated, the model samples the state $\mathtt{emacs}(\mathtt{hmm1},\mathtt{tex})$ from $G_{\Sigma_2}$ using $\mu$. As indicated by the dashed arrow, $\mathtt{emacs}(\mathtt{F},\mathtt{tex})$ is more specific than $\mathtt{emacs}(\mathtt{F},\mathtt{U})$. Moreover, $\mathtt{emacs}(\mathtt{hmm1},\mathtt{tex})$ matches $\mathtt{emacs}(\mathtt{F},\mathtt{tex})$. Thus, the model enters $\mathtt{emacs}(\mathtt{F},\mathtt{tex})$. Since the value of `F` was already instantiated in the previous abstract state, $\mathtt{emacs}(\mathtt{hmm1},\mathtt{tex})$ is sampled with probability 1.0. Now, the model goes over to $\mathtt{latex}(\mathtt{F},\mathtt{tex})$, emitting $\mathtt{emacs}(\mathtt{hmm1})$ because the abstract observation $\mathtt{emacs}(\mathtt{F})$ is already fully instantiated. Again, since $F$ was already instantiated, $\mathtt{latex}(\mathtt{hmm1},\mathtt{tex})$ is sampled with probability 1.0. Next, we move on to $\mathtt{emacs}(\mathtt{F}',\mathtt{U})$, emitting $\mathtt{latex}(\mathtt{hmm1})$. Variables $\mathtt{F}'$ and `U` in $\mathtt{emacs}(\mathtt{F}',\mathtt{U})$ were not yet bound; so, values, say `lohmm1` and `others`, are sampled from $\mu$. The dotted arrow brings us back to $\mathtt{emacs}(\mathtt{F},\mathtt{U})$. Because variables are implicitly universally quantified in abstract transitions, the scope of variables is restricted to single abstract transitions. In turn, $F$ is treated as a distinct, new variable, and is automatically unified with $\mathtt{F}'$, which is bound to `lohmm1`. In contrast, variable `U` is already instantiated. Emitting $\mathtt{emacs}(\mathtt{lohmm1})$,

the model makes a transition to $\mathtt{ls}(\mathtt{U}')$. Assume that it samples `tex` for $\mathtt{U}'$. Then, it remains in $\mathtt{ls}(\mathtt{U}')$ with probability 0.4 . Considering all possible samples, allows one to prove the following theorem.

**Theorem 5.14** (Semantics) *A logical hidden Markov model over a language $\Sigma$ defines a discrete time stochastic process, i.e., a sequence of random variables $\langle X_t \rangle_{t=1,2,\ldots}$, where the domain of $X_t$ is $\mathrm{hb}(\Sigma) \times \mathrm{hb}(\Sigma)$. The induced probability measure over the Cartesian product $\bigotimes_t \mathrm{hb}(\Sigma) \times \mathrm{hb}(\Sigma)$ exists and is unique for each $t > 0$ and in the limit $t \to \infty$.* ∘

*Proof sketch:* Let $M = (\Sigma, \mu, \Delta, \Upsilon)$ be a logical hidden Markov model. To show that $M$ specifies a time discrete stochastic process, i.e., a sequence of random variables $\langle X_t \rangle_{t=1,2,\ldots}$, where the domains of the random variable $X_t$ is $\mathrm{hb}(\Sigma)$, the Herbrand base over $\Sigma$, we define the *immediate state operator* $T_M$-operator and the *current emission operator* $E_M$-operator.

**Definition 5.15** ($T_M$-Operator, $E_M$-Operator ) The operators $T_M : 2^{\mathrm{hb}_\Sigma} \to 2^{\mathrm{hb}_\Sigma}$ and $E_M : 2^{\mathrm{hb}_\Sigma} \to 2^{\mathrm{hb}_\Sigma}$ are

$$T_M(I) = \{H\sigma_B\sigma_H \mid \exists(p : H \xleftarrow{O} B) \in M : B\sigma_B \in I, H\sigma_B\sigma_H \in G_\Sigma(H)\}$$

$$E_M(I) = \{O\sigma_B\sigma_H\sigma_O \mid \exists(p : H \xleftarrow{O} B) \in M : B\sigma_B \in I,\ H\sigma_B\sigma_G \in G_\Sigma(H) \text{ and } O\sigma_B\sigma_H\sigma_O \in G_\Sigma(O)\}$$

∘

For each $i = 1, 2, 3, \ldots$, the set $T_M^{i+1}(\{\mathtt{start}\}) := T_M(T_M^i(\{\mathtt{start}\}))$ with $T_M^1(\{\mathtt{start}\}) := T_M(\{\mathtt{start}\})$ specifies the state set at clock $i$, which forms a random variable $Y_i$. The set $U_M^i(\{\mathtt{start}\})$ specifies the possible symbols emitted when transitioning from $i$ to $i+1$. It forms the variable $U_i$. Each $Y_i$ (resp. $U_i$) can be extended to a random variable $Z_i$ (resp. $U_i$) over $\mathrm{hb}_\Sigma$:

$$P(Z_i = z) = \begin{cases} 0.0 : z \notin T_M^i(\{\mathtt{start}\}) \\ P(Y_i = z) : \text{otherwise} \end{cases}$$

Figure 5.3 depicts the influence relation among $Z_i$ and $U_i$. Using standard arguments from probability theory and noting that

$$P(U_i = U_i \mid Z_{i+1} = z_{i+1}, Z_i = z_i) = \frac{P(Z_{i+1} = z_{i+1}, U_i = u_i \mid Z_i)}{\sum_{u_i} P(Z_{i+1}, u_i \mid Z_i)}$$

$$\text{and } P(Z_{i+1} \mid Z_i) = \sum_{u_i} P(Z_{i+1}, u_i \mid Z_i)$$

where the probability distributions are due to equation (5.1), it is easy to show that Kolmogorov's extension theorem (see [Bauer, 1991, Fristedt and Gray, 1997]) holds. Thus, $M$ specifies a unique probability distribution over $\bigotimes_{i=1}^t (Z_i \times U_i)$ for each $t > 0$ and in the limit $t \to \infty$. □

This is akin to unrolling recurrent neural networks [Dean and Kanazawa, 1988] and dynamic Bayesian networks [Williams and Zipser, 1995], and to grounding clause programs [Lloyd, 1989].

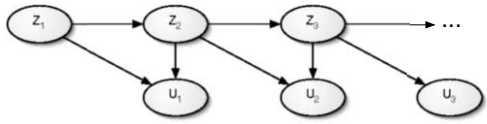



**Figure 5.3.** Discrete time stochastic process induced by a logical hidden Markov model. The nodes $Z_i$ and $U_i$ represent random variables over $\mathrm{hb}_\Sigma$.

## 5.3 Design Choices

Before going on to the basic inference tasks, let us address some design choices underlying logical hidden Markov models: the Mealy representation of abstract transitions, the naïve Bayes factorization of the selection distribution, and the focus on distributions over sequences of fixed length.

### 5.3.1 Mealy Representation

Logical hidden Markov models have been introduced as Mealy machines, i.e., output symbols are associated with transitions.

Mealy machines fit our logical setting quite intuitively as they directly encode the conditional probability $P(\mathtt{O}, \mathtt{S}' | \mathtt{S})$ of making a transition from $\mathtt{S}$ to $\mathtt{S}'$ emitting an observation $\mathtt{O}$. Logical hidden Markov models define this distribution as

$$P(\mathtt{O}, \mathtt{S}' | \mathtt{S}) = \sum\nolimits_{p:\mathtt{H} \xleftarrow{\mathtt{O}'} \mathtt{B}} p \cdot \mu(\mathtt{S}' \mid \mathtt{H}\sigma_\mathtt{B}) \cdot \mu(\mathtt{O} \mid \mathtt{O}'\sigma_\mathtt{B}\sigma_\mathtt{H})$$

where the sum runs over all abstract transitions $\mathtt{H} \xleftarrow{\mathtt{O}'} \mathtt{B}$ such that $\mathtt{B}$ is most specific for $\mathtt{S}$. Observations correspond to (partially) observed proof steps and, hence, provide information shared among heads and bodies of abstract transitions. In contrast, HMMs are usually introduced as *Moore* machines. Here, output symbols are associated with states implicitly assuming $\mathtt{O}$ and $\mathtt{S}'$ to be independent. Thus, $P(\mathtt{O}, \mathtt{S}' \mid \mathtt{S})$ factorizes into $P(\mathtt{O} \mid \mathtt{S}) \cdot P(\mathtt{S}' \mid \mathtt{S})$. This makes it more difficult to observe information shared among heads and bodies. In turn, Moore-LOHMMs are less intuitive and harder to understand. More precisely, it can essentially be shown that — as in the propositional case — Mealy- and Moore-*logical hidden Markov models* are equivalent.

To see this, let $L$ be a Mealy-LOHMM according to definition 5.13. In the following, we will derive the notation of an equivalent logical hidden Markov model $L'$ in Moore representation where there are abstract transitions and abstract emissions (see below). Each predicate $\mathtt{b}/n$ in $L$ is extended to $\mathtt{b}/n+1$ in $L'$. The domains of the first $n$ arguments are the same as for $\mathtt{b}/n$. The last argument will store the observation to be emitted. More precisely, for each abstract transition

$$p : \mathtt{h}(\mathtt{w}_1, \ldots, \mathtt{w}_l) \xleftarrow{\mathtt{o}(\mathtt{v}_1, \ldots, \mathtt{v}_k)} \mathtt{b}(\mathtt{u}_1, \ldots, \mathtt{u}_n)$$

in $L$, there is an abstract transition

$$p : \mathtt{h}(\mathtt{w}_1, \ldots, \mathtt{w}_l, \mathtt{o}(\mathtt{v}'_1, \ldots, \mathtt{v}'_k)) \leftarrow \mathtt{b}(\mathtt{u}_1, \ldots, \mathtt{u}_n, \_)$$

in $L'$. The primes in $\mathtt{o}(\mathtt{v}'_1,\ldots,\mathtt{v}'_k)$ denote that we replaced each free [23] variables $\mathtt{o}(\mathtt{v}_1,\ldots,\mathtt{v}_k)$ by some distinguished constant symbol, say #. Due to this, it holds that

$$\mu(\mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l)) = \mu(\mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l,\mathtt{o}(\mathtt{v}'_1,\ldots,\mathtt{v}'_k)))\ , \tag{5.3}$$

and $L'$'s output distribution can be specified using *abstract emissions*, which are expressions of the form

$$1.0 : \mathtt{o}(\mathtt{v}_1,\ldots,\mathtt{v}_k) \leftarrow \mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l,\mathtt{o}(\mathtt{v}'_1,\ldots,\mathtt{v}'_k))\ . \tag{5.4}$$

The semantics of an abstract transition in $L'$ is that being in some state $\mathtt{S}'_t \in G_{\Sigma'}(\mathtt{b}(\mathtt{u}_1,\ldots,\mathtt{u}_n,\_))$ the system will make a transition into state $\mathtt{S}'_{t+1} \in G_{\Sigma'}(\mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l,\mathtt{o}(\mathtt{v}'_1,\ldots,\mathtt{v}'_k)))$ with probability

$$p \cdot \mu(\mathtt{S}'_{t+1} \mid \mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l,\mathtt{o}(\mathtt{v}'_1,\ldots,\mathtt{v}'_k)) \mid \sigma_{\mathtt{S}'_t}) \tag{5.5}$$

where $\sigma_{\mathtt{S}'_t} = \mathrm{mgu}(\mathtt{S}'_t, \mathtt{b}(\mathtt{u}_1,\ldots,\mathtt{u}_n,\_))$. Due to Equation (5.3), Equation (5.5) can be rewritten as $p \cdot \mu(\mathtt{S}'_{t+1} \mid \mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l) \mid \sigma_{\mathtt{S}'_t})$ . Due to equation (5.4), the system will emit the output symbol $\mathtt{o}_{t+1} \in G_{\Sigma'}(\mathtt{o}(\mathtt{v}_1,\ldots,\mathtt{v}_k))$ in state $\mathtt{S}'_{t+1}$ with probability $\mu(\mathtt{o}_{t+1} \mid \mathtt{o}(\mathtt{v}_1,\ldots,\mathtt{v}_k)\sigma_{\mathtt{S}'_{t+1}}\sigma_{\mathtt{S}'_t})$ where $\sigma_{\mathtt{S}'_{t+1}} = \mathrm{mgu}(\mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l,\mathtt{o}(\mathtt{v}'_1,\ldots,\mathtt{v}'_k)),\mathtt{S}'_{t+1})$. Due to the construction of $L'$, there exists a triple $(\mathtt{S}_t,\mathtt{S}_{t+1},\mathtt{O}_{t+1})$ in $L$ for each triple $(\mathtt{S}'_t,\mathtt{S}'_{t+1},\mathtt{O}_{t+1})$, $t > 0$, in $L'$ (and vise versa). Hence,both logical hidden Markov models assign the same overall transition probability.

$L$ and $L'$ differ only in the way the initialize sequences $\langle(\mathtt{S}'_t,\mathtt{S}'_{t+1},\mathtt{O}_{t+1}\rangle_{t=0,2\ldots,T}$ (resp. $\langle(\mathtt{S}_t,\mathtt{S}_{t+1},\mathtt{O}_{t+1}\rangle_{t=0,2\ldots,T}$). Whereas $L$ starts in some state $\mathtt{S}_0$ and makes a transition to $\mathtt{S}_1$ emitting $\mathtt{O}_1$, the Moore-LOHMM $L'$ is supposed to emit a symbol $\mathtt{O}_0$ in $\mathtt{S}'_0$ before making a transition to $\mathtt{S}'_1$. We compensate for this using the prior distribution. The existence of the correct prior distribution for $L'$ can be seen as follows. In $L$, there are only finitely many states reachable at time $t = 1$, i.e, $P_L(q_0 = \mathtt{S}) > 0$ holds for only a finite set of ground states $\mathtt{S}$. The probability $P_L(q_0 = \mathtt{s})$ can be computed similar to $\alpha_1(\mathtt{S})$. We set $t = 1$ in line 6, neglecting the condition on $\mathtt{O}_{t-1}$ in line 10, and dropping $\mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_B\sigma_H)$ from line 14. Completely listing all states $\mathtt{S} \in S_1$ together with $P_L(q_0 = \mathtt{S})$, i.e., $P_L(q_0 = \mathtt{S}) : \mathtt{S} \leftarrow \mathtt{start}$ , constitutes the prior distribution of $L'$.

The argumentation basically followed the approach to transform a Mealy machine into a Moore machine, see e.g. [Hopcroft and Ullman, 1979]. Furthermore, the mapping of a Moore-LOHMM — as introduced in the present section — into a Mealy-LOHMM is straightforward.

### 5.3.2 Naïve Bayes Selection Distribution

The naïve Bayes approach for the selection distribution reduces the model complexity at the expense of a lower expressivity: functors are neglected and variables are treated independently. Adapting more expressive approaches is an interesting future line of research. For instance, *Bayesian networks* allow one to represent *factorial hidden*

[23] A variable $\mathtt{X} \in \mathrm{vars}(\mathtt{o}(\mathtt{v}_1,\ldots,\mathtt{v}_k))$ is free iff $\mathtt{X} \notin \mathrm{vars}(\mathtt{h}(\mathtt{w}_1,\ldots,\mathtt{w}_l)) \cup \mathrm{vars}(\mathtt{b}(\mathtt{u}_1,\ldots,\mathtt{u}_n))$.

*Markov models* [Ghahramani and Jordan, 1997]. Factorial hidden Markov models can be viewed as a special type of logical hidden Markov models, where the hidden states are summarized by a $2 \cdot k$-ary abstract state. The first $k$ arguments encode the $k$ state variables, and the last $k$ arguments serve as a memory of the previous joint state. $\mu$ of the $i$-th argument is conditioned on the $i + k$-th argument. *Markov chains* allow one to sample compound terms of variable, finite depth such as `s(s(s(0)))` and to model e.g. misspelled filenames. This is akin to the idea of *generalized hidden Markov models* [Kulp et al., 1996] in which each node may output a finite sequence of symbols rather than a single symbol.

### 5.3.3 Distribution over Fixed Length Sequences

Logical hidden Markov models — as introduced here — specify a probability distribution over all sequences of a given length. Reconsider the logical hidden Markov model in Figure 5.1. Already the probabilities of all observation sequences of length 1, i.e., `ls`, `emacs(hmm1)`, and `emacs(lohmm1)`) sum up to 1. More precisely, for each $t > 0$ it holds that

$$\sum_{x_1,\ldots,x_t} P(X_1 = x_1, \ldots, X_t = x_t) = 1.0 \; .$$

In order to model a distribution over sequences of variable length, i.e.,

$$\sum_{t>0} \sum_{x_1,\ldots,x_t} P(X_1 = x_1, \ldots, X_t = x_t) = 1.0$$

we may add a distinguished `end` state. The `end` state is absorbing in that whenever the model makes a transition into this state, it terminates the observation sequence generated.

§ 6

## Three Basic Inference Problems for Logical HMMs *

*... in which the basic inference algorithms for hidden Markov models are upgraded for use in logical hidden Markov models, the benefits of logical hidden Markov models are investigated, and logical hidden Markov models are applied to real world data ...*

As for HMMs, three inference problems are of interest. Let $M$ be a logical hidden Markov model and let $\mathsf{O} = \mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_T$, $T > 0$, be a finite sequence of ground observations:

**(1) Evaluation:** Determine the probability $P(\mathsf{O} \mid M)$ that sequence $\mathsf{O}$ was generated by the model $M$.

* Builds on [Kersting et al., 2002, Raiko et al., 2002, Kersting et al., 2003b, 2006].

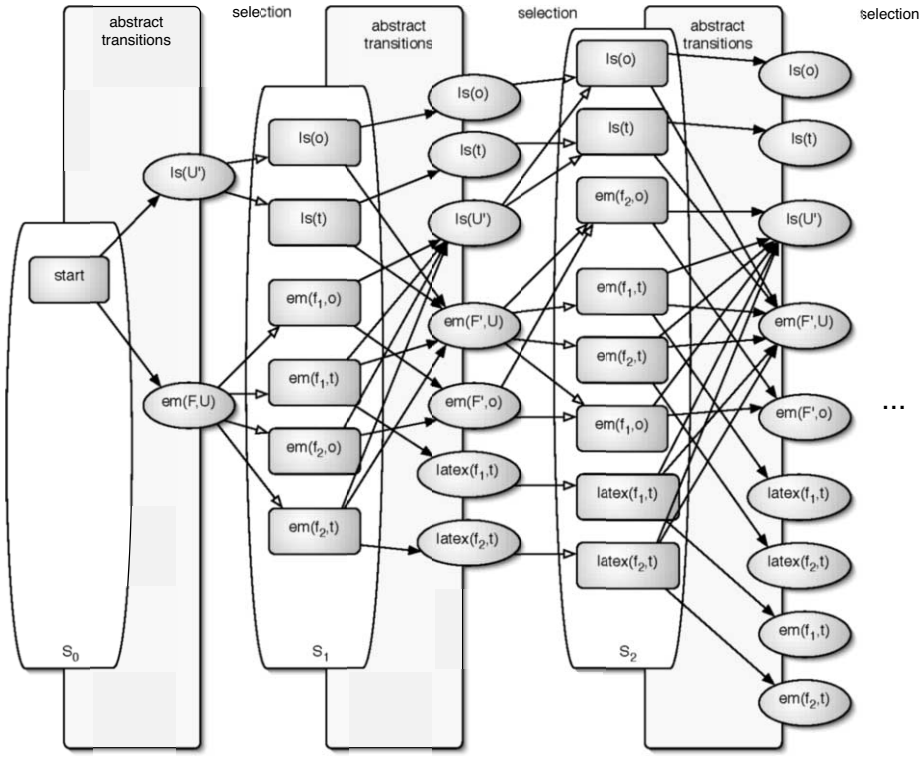



**Figure 6.1.** Trellis induced by the logical hidden Markov model in Figure 5.1. The sets of reachable states at time $0, 1, \ldots$ are denoted by $S_0, S_1, \ldots$ In contrast with HMMs, there is an additional layer where the states are sampled from abstract states.

**(2) Most likely state sequence:** Determine the hidden state sequence $\mathsf{S}^*$ that has most likely produced the observation sequence $\mathsf{O}$, i.e. $\mathsf{S}^* = \arg\max_{\mathsf{S}} P(\mathsf{S} \mid \mathsf{O}, M)$ .

**(3) Parameter estimation:** Given a set $\mathbf{O} = \{\mathsf{O}_1, \ldots, \mathsf{O}_k\}$ of observation sequences, determine the most likely parameters $\boldsymbol{\lambda}^*$ for the abstract transitions and the selection distribution of $M$, i.e. $\lambda^* = \arg\max_{\lambda} P(\mathbf{O} \mid \boldsymbol{\lambda})$ .

We will now address each of these problems in turn by upgrading the existing solutions for HMMs. This will be realized by computing a grounded trellis as in Figure 6.1. The possible ground successor states of any given state are computed by first selecting the applicable abstract transitions and then applying the selection probabilities (while taking into account the substitutions) to ground the resulting states. This two-step factorization is coalesced into one step for HMMs.

**Algorithm II.1**: *Forward Procedure*: A dynamic programming approach for computing the forward probability $\alpha_t(\mathtt{S})$ for a given logical hidden Markov models $M = (\Sigma, \mu, \Delta)$. We assume $\mathtt{O} \equiv \mathtt{start}$ for each abstract transition $p : \mathtt{H} \leftarrow \mathtt{start} \in \Upsilon$. Boxed parts specify the major differences to the HMM formula: unification and $\mu$ are taken into account.

$S_0 := \{\mathtt{start}\}$
**for** $t = 1, 2, \ldots, T$ **do**
  $S_t = \emptyset$ /* *initialize the set of reachable states at clock* $t$ */
  **foreach** $\mathtt{S} \in S_{t-1}$ **do**
    **foreach** $\boxed{\textit{maximally specific } p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta \textit{ s.t. } \sigma_{\mathtt{B}} = \mathrm{mgu}(\mathtt{S}, \mathtt{B}) \textit{ exists}}$ **do**
      **foreach** $\boxed{\mathtt{S}' = \mathtt{H}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}} \in G_\Sigma(\mathtt{H}\sigma_{\mathtt{B}}) \textit{ s.t. } \mathtt{O}_{t-1} \textit{ unifies with } \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}}}$ **do**
        **if** $\mathtt{S}' \notin S_t$ **then**
          $S_t := S_t \cup \{\mathtt{S}'\}$
          $\alpha_t(\mathtt{S}') := 0.0$
        $\alpha_t(\mathtt{S}') := \alpha_t(\mathtt{S}') + \alpha_{t-1}(\mathtt{S}) \cdot p \cdot \boxed{\mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})}$
**Return** $P(\mathtt{O} \mid M) = \sum_{\mathtt{S} \in S_T} \alpha_T(\mathtt{S})$

## 6.1 Evaluation

To evaluate $\mathtt{O}$, consider the probability of the partial observation sequence $\mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_t$ and (ground) state $\mathtt{S}$ at time $t$, $0 < t \leq T$, given the model $M = (\Sigma, \mu, \Delta, \Upsilon)$

$$\alpha_t(\mathtt{S}) := P(\mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_t, q_t = \mathtt{S} \mid M)$$

where $q_t = \mathtt{S}$ denotes that the system is in state $\mathtt{S}$ at time $t$. As for HMMs, $\alpha_t(\mathtt{S})$ can be computed using a dynamic programming approach. For $t = 0$, we set $\alpha_0(\mathtt{S}) = P(q_0 = \mathtt{S} \mid M)$ , i.e., $\alpha_0(\mathtt{S})$ is the probability of starting in state $\mathtt{S}$ and, for $t > 0$, we compute $\alpha_t(\mathtt{S})$ based on $\alpha_{t-1}(\mathtt{S}')$. The computations are summarized in the *forward procedure* in Algorithm II.1 where we assume for the sake of simplicity $\mathtt{O} \equiv \mathtt{start}$ for each abstract transition $p : \mathtt{H} \leftarrow \mathtt{start} \in \Upsilon$. Furthermore, the boxed parts specify the major differences to the HMM formula: unification and $\mu$ are taken into account. Clearly, as for HMMs

$$P(\mathtt{O} \mid M) = \sum\nolimits_{\mathtt{S} \in S_T} \alpha_T(\mathtt{S})$$

holds.

The computational complexity of this *forward procedure* is $\mathcal{O}(T \cdot s \cdot (|\mathbf{B}| + o \cdot g)) = \mathcal{O}(T \cdot s^2)$ where $s = \max_{t=1,2,\ldots,T} |S_t|$ , $o$ is the maximal number of outgoing abstract transitions with regard to an abstract state, and

$g$ is the maximal number of ground instances of an abstract state. In a completely analogous manner, one can devise a *backward procedure* to compute

$$\beta_t(\mathtt{S}) = P(\mathtt{O}_{t+1}, \mathtt{O}_{t+2}, \ldots, \mathtt{O}_T \mid q_t = \mathtt{S}, M) \; .$$

This will be useful for solving Problem **(3)**.

## 6.2 Most Likely State Sequences

Having a forward procedure, it is straightforward to adapt the Viterbi algorithm as a solution to Problem **(2)**, i.e., for computing the most likely state sequence. Let $\delta_t(\mathtt{S})$ denote the highest probability along a single path at time $t$, which accounts for the first $t$ observations and ends in state $\mathtt{S}$, i.e.,

$$\delta_t(\mathtt{S}) = \max_{\mathtt{S}_0, \mathtt{S}_1, \ldots, \mathtt{S}_{t-1}} P(\mathtt{S}_0, \mathtt{S}_1, \ldots, \mathtt{S}_{t-1}, \mathtt{S}_t = \mathtt{S}, O_1, \ldots, O_{t-1} | M) \; .$$

The procedure for finding the most likely state sequence basically follows the *forward procedure*, Algorithm II.1. Instead of summing over all ground transition probabilities in line 10, we maximize over them, cf. Algorithm II.2. Here, $\delta_t(\mathtt{S}, \mathtt{S}')$ stores the probability of making a transition from $\mathtt{S}$ to $\mathtt{S}'$ and $\psi_t(\mathtt{S}')$ (with $\psi_1(S) = \mathtt{start}$ for all states $S$) keeps track of the state maximizing the probability along a single path at time $t$, which accounts for the first $t$ observations and ends in state $\mathtt{S}'$. The most likely hidden state sequence $\mathtt{S}^*$ can now be computed as

$$\begin{aligned} &\mathtt{S}^*_{T+1} = \arg \max_{\mathtt{S} \in S_{T+1}} \delta_{T+1}(\mathtt{S}) \\ &\text{and } \mathtt{S}^*_t = \psi_t(\mathtt{S}^*_{t+1}) \text{ for } t = T, T-1, \ldots, 1 \; . \end{aligned}$$

One can also consider problem **(2)** on a more abstract level. Instead of considering all contributions of different abstract transitions $\mathtt{T}$ to a single ground transition from state $\mathtt{S}$ to state $\mathtt{S}'$ in line 10, one might also consider the most likely abstract transition only. This is realized by replacing line 10 in the forward procedure with

$$\alpha_t(\mathtt{S}') := \max(\alpha_t(\mathtt{S}'), \alpha_{t-1}(\mathtt{S}) \cdot p \cdot \; \mu(\mathtt{S}' \mid \mathtt{H}\sigma_\mathtt{B}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_\mathtt{B}\sigma_\mathtt{H})) \; .$$

This solves the problem of finding the **(2′) most likely state and abstract transition sequence**:

> Determine the sequence of states and abstract transitions $\mathtt{GT}^* = S_0, \mathtt{T}_0, S_1, \mathtt{T}_1, \mathtt{S}_2, \ldots, \mathtt{S}_\mathtt{T}, \mathtt{T}_T, \mathtt{S}_{\mathtt{T}+1}$ where there exists substitutions $\theta_i$ with $S_{i+1} \leftarrow S_i \equiv \mathtt{T}_i \, \theta_i$ that has most likely produced the observation sequence $\mathtt{O}$, i.e. $\mathtt{GT}^* = \arg\max_\mathtt{GT} P(\mathtt{GT} \mid \mathtt{O}, M)$ .

Thus, logical hidden Markov models also pose new types of inference problems.

**Algorithm II.2**: *Viterbi Algorithm*: A dynamic programming approach for computing the hidden state sequence $\mathtt{S}^*$ that has most likely produced the observation sequence $\mathtt{O}$ for a given logical hidden Markov models $M = (\Sigma, \mu, \Delta)$. $\delta_t(\mathtt{S}, \mathtt{S}')$ stores the probability of making a transition from $\mathtt{S}$ to $\mathtt{S}'$ and $\psi_t(\mathtt{S}')$ (with $\psi_1(S) = \mathtt{start}$ for all states $S$) keeps track of the state maximizing the probability along a single path at time $t$, which accounts for the first $t$ observations and ends in state $\mathtt{S}'$. We assume $\mathtt{O} \equiv \mathtt{start}$ for each abstract transition $p : \mathtt{H} \leftarrow \mathtt{start} \in \Upsilon$.

$S_0 := \{\mathtt{start}\}$
**for** $t = 1, 2, \ldots, T$ **do**
  $S_t = \emptyset$ /* *initialize the set of reachable states at clock* $t$ */
  **foreach** $\mathtt{S} \in S_{t-1}$ **do**
    **foreach** *maximally specific* $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta$ *s.t.*$\sigma_{\mathtt{B}} = \mathrm{mgu}(\mathtt{S}, \mathtt{B})$ *exists* **do**
      **foreach** $\mathtt{S}' = \mathtt{H}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}} \in G_\Sigma(\mathtt{H}\sigma_{\mathtt{B}})$ *s.t.* $\mathtt{O}_{t-1}$ *unifies with* $\mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}}$ **do**
        **if** $\mathtt{S}' \notin S_t$ **then**
          $S_t := S_t \cup \{\mathtt{S}'\}$
          $\delta_t(\mathtt{S}, \mathtt{S}') := 0.0$
        $\delta_t(\mathtt{S}, \mathtt{S}') := \delta_t(\mathtt{S}, \mathtt{S}') + \delta_{t-1}(\mathtt{S}) \cdot p \cdot \mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})$
  **foreach** $\mathtt{S}' \in S_t$ **do**
    $\delta_t(\mathtt{S}') = \max_{\mathtt{S} \in S_{t-1}} \delta_t(\mathtt{S}, \mathtt{S}')$
    $\psi_t(\mathtt{S}') = \arg\max_{\mathtt{S} \in S_{t-1}} \psi_t(\mathtt{S}, \mathtt{S}')$
**Return** all $\delta_t(\mathtt{S}')$ and $\psi_t(\mathtt{S}')$

## 6.3 Parameter Estimation

For parameter estimation, we have to estimate the maximum likelihood transition probabilities and selection distributions. To estimate the former, we upgrade the well-known *Baum-Welch* algorithm [Baum, 1972] for estimating the maximum likelihood parameters of HMMs and probabilistic context-free grammars.

For HMMs, the Baum-Welch algorithm computes the improved estimate $\overline{p}$ of the transition probability of some (ground) transition $\mathtt{T} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ by taking the ratio

$$\overline{p} = \frac{\xi(\mathtt{T})}{\sum_{\mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B} \in \Delta \cup \Upsilon} \xi(\mathtt{T}')} \tag{6.1}$$

between the expected number $\xi(\mathtt{T})$ of times of making the transitions $\mathtt{T}$ at any time given the model $M$ and an observation sequence $\mathtt{O}$, and the total number of times a transitions is made from $\mathtt{B}$ at any time given $M$ and $\mathtt{O}$.

Basically the same applies when $\mathtt{T}$ is an abstract transition. However, we have to be a little bit more careful because we have no direct access to $\xi(\mathtt{T})$. Let $\xi_t(\mathrm{gcl}, \mathtt{T})$ be

**Algorithm II.3**: The adapted Baum-Welch algorithm for re-estimating the probabilities of abstract transitions.

```
1  /* initialization of expected counts */
2  foreach T ∈ Δ do
3  |  ξ(T) := m   /* or 0 if not using pseudocounts */
4  /* compute expected counts */
5  for t = 0, 1, ..., T do
6  |  foreach S ∈ S_t do
7  |  |  foreach max. specific T ≡ p : H ←0– B ∈ Δ s.t. σ_B = mgu(S, B) exists do
8  |  |  |  foreach S' = Hσ_Bσ_H ∈ G_Σ(Hσ_B) s.t. S' ∈ S_{t+1} ∧ mgu(O_t, Oσ_Bσ_H) exists do
9  |  |  |  |  ξ(T) := ξ(T) + α_t(S) · p · β_{t+1}(S') / P(O | M) ·
   |  |  |  |  [ μ(S' | Hσ_B) · μ(O_{t−1} | Oσ_Bσ_H) ]
10 |  |  |  |
11 |  |  |
12 |  |
13
```

the probability of following the abstract transition $\mathtt{T}$ via its ground instance $\mathrm{gcl} \equiv p : \mathtt{GH} \xleftarrow{\mathtt{GO}} \mathtt{GB}$ at time $t$, i.e.,

$$\xi_t(\mathrm{gcl}, \mathtt{T}) = \frac{\alpha_t(\mathtt{GB}) \cdot p \cdot \beta_{t+1}(\mathtt{GH})}{P(\mathsf{O} \mid M)} \cdot \boxed{\mu(\mathtt{GH} \mid \mathtt{H}\sigma_\mathtt{B}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_\mathtt{B}\sigma_\mathtt{H})}\,, \tag{6.2}$$

where $\sigma_\mathtt{B}, \sigma_\mathtt{H}$ are as in the forward procedure (see above) and $P(\mathsf{O} \mid M)$ is the probability that the model generated the sequence $\mathsf{O}$. Again, the boxed terms constitute the main difference to the corresponding HMM formula. In order to apply Equation (6.1) to compute improved estimates of probabilities associated with abstract transitions, we set

$$\xi(\mathtt{T}) = \sum_{t=1}^{T} \xi_t(\mathtt{T}) = \sum_{t=1}^{T} \sum_{\mathrm{gcl}} \xi_t(\mathrm{gcl}, \mathtt{T})$$

where the inner sum runs over all ground instances of $\mathtt{T}$.

This leads to re-estimation method in Algorithm II.3, where we assume that the sets $S_i$ of reachable states are reused from the computations of the $\alpha$- and $\beta$-values. In Algorithm II.3, Equation (6.2) can be found in line 7. In line 3, we set pseudocounts as small sample-size regularizers. Other methods to avoid a biased underestimate of probabilities and even zero probabilities such as $m$-estimates, see e.g. Mitchell [1997], can be easily adapted.

To estimate the selection probabilities, recall that $\mu$ follows a naïve Bayes scheme. Therefore, the estimated probability for a domain element $d \in D$ for some domain $D$ is the ratio between the number of times $d$ is selected and the number of times any $d' \in D$ is selected. The procedure for computing the $\xi$-values can thus be reused.

Altogether, the Baum-Welch algorithm works as follows: While not converged, estimate

(1) the abstract transition probabilities, and
(2) the selection probabilities.

Since it is an instance of the EM algorithm, it increases the likelihood of the data with every update, and according to McLachlan and Krishnan [1997], it is guaranteed to reach a stationary point. All standard techniques to overcome limitations of EM algorithms are applicable. The computational complexity (per iteration) is $\mathcal{O}(k \cdot (\alpha + d)) = \mathcal{O}(k \cdot T \cdot s^2 + k \cdot d)$ where $k$ is the number of sequences, $\alpha$ is the complexity of computing the $\alpha$-values (see above), and $d$ is the sum over the sizes of domains associated to predicates. In Chapter 7, we will show how to combined the Baum-Welch algorithm with structure search for model selection of logical hidden Markov models using *inductive logic programming* refinement operators. The refinement operators account for different abstraction levels, which have to be explored.

## 6.4 Advantages of Logical Hidden Markov Models

In this section, we will investigate the benefits of logical hidden Markov models:

**(1)** logical hidden Markov models are strictly more expressive than HMMs, and,
**(2)** using abstraction, logical variables and unification can be beneficial.

More specifically, with **(2)**, we will show that

**(B1)** logical hidden Markov models can be — by design — smaller than their propositional instantiations, and
**(B2)** unification can yield better log-likelihood estimates.

### 6.4.1 On the Expressivity of Logical Hidden Markov Models

Whereas HMMs specify probability distributions over regular languages, logical hidden Markov models specify probability distributions over more expressive languages.

**Theorem 6.1** *For any (consistent) probabilistic context-free grammar (PCFG) $G$ for some language $\mathcal{L}$ there exists a logical hidden Markov model $M$ s.t. $P_G(w) = P_M(w)$ for all $w \in \mathcal{L}$.* ∘

*Proof sketch:* Let $T$ be a terminal alphabet and $N$ a nonterminal alphabet. A *probabilistic context-free grammar* (PCFG) $G$ consists of a distinguished start symbol $S \in N$ plus a finite set of productions of the form $p : X \to \alpha$, where $X \in N$, $\alpha \in (N \cup T)^*$ and $p \in [0, 1]$. For all $X \in N$, $\sum_{:X \to \alpha} p = 1$. A PCFG defines a stochastic process with sentential forms as states, and leftmost rewriting steps as transitions. We denote a single rewriting operation of the grammar by a single arrow $\to$. If as a result of one ore more rewriting operations we are able to rewrite $\beta \in (N \cup T)^*$ as a sequence $\gamma \in (N \cup T)^*$ of nonterminals and terminals, then we write $\beta \Rightarrow^* \gamma$. The probability of this rewriting is the product of all probability values associated to productions used in the derivation. We assume $G$ to be consistent, i.e., that the sum of all probabilities of derivations $S \Rightarrow^* \beta$ such that $\beta \in T^*$ sum to 1.0.

We can assume that the PCFG $G$ is in Greibach normal form. This follows from Abney et al. [1999]'s Theorem 6 because $G$ is consistent. Thus, every production $P \in G$ is of the form $p : X \to aY_1 \ldots Y_n$ for some $n \geq 0$. In order to encode $G$ as a logical hidden Markov model $M$, we introduce (1) for each non-terminal symbol $X$ in $G$ a constant symbol $\mathtt{nX}$ and (2) for each terminal symbol $t$ in $G$ a constant symbol $\mathtt{t}$. For each production $P \in G$, we include an abstract transition of the form $p : \mathtt{stack}([\mathtt{nY_1}, \ldots, \mathtt{nY_n}|\mathtt{S}]) \overset{\mathtt{a}}{\leftarrow} \mathtt{stack}([\mathtt{nX}|\mathtt{S}])$, if $n > 0$, and $p : \mathtt{stack}(\mathtt{S}) \overset{\mathtt{a}}{\leftarrow} \mathtt{stack}([\mathtt{nX}|\mathtt{S}])$, if $n = 0$. Furthermore, we include $1.0 : \mathtt{stack}([\mathtt{s}]) \leftarrow \mathtt{start}$ and $1.0 : \mathtt{end} \overset{\mathtt{end}}{\longleftarrow} \mathtt{stack}([])$. It is now straightforward to prove by induction that $M$ and $G$ are equivalent. □

The proof makes use of functors. Without functors, logical hidden Markov models cannot encode PCFGs. In this case, because the Herbrand base is finite, it can be proven that there always exists an equivalent HMM.

Moreover, if functors are allowed, logical hidden Markov models are strictly more expressive than probabilistic context-free grammars (PCFGs). They can specify probability distributions over some languages that are context-sensitive

**Example 6.2** The logical hidden Markov model

$$
\begin{array}{rrcl}
1.0: & \mathtt{stack(s(0),s(0))} & \leftarrow & \mathtt{start} \\
0.8: & \mathtt{stack(s(X),s(X))} & \overset{\mathtt{a}}{\leftarrow} & \mathtt{stack(X,X)} \\
0.2: & \mathtt{unstack(s(X),s(X))} & \overset{\mathtt{a}}{\leftarrow} & \mathtt{stack(X,X)} \\
1.0: & \mathtt{unstack(X,Y)} & \overset{\mathtt{b}}{\leftarrow} & \mathtt{unstack(s(X),Y)} \\
1.0: & \mathtt{unstack(s(0),Y)} & \overset{\mathtt{c}}{\leftarrow} & \mathtt{unstack(s(0),s(Y))} \\
1.0: & \mathtt{end} & \overset{\mathtt{end}}{\longleftarrow} & \mathtt{unstack(s(0),s(0))}
\end{array}
$$

defines a distribution over $\{a^n b^n c^n \mid n > 0\}$. ○

Finally, the use of logical variables also enables one to deal with *identifiers*. Identifiers are special types of constants that denote objects. Indeed, recall the UNIX command sequence `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, ... from Example 5.1. The filename `lohmms.tex` is an identifier. Usually, the specific identifiers do not matter but rather the fact that the same object occurs multiple times in the sequence. Logical hidden Markov models can easily deal with identifiers by setting the selection distribution $\mu$ to 1 for the arguments in which identifiers can occur. Unification then takes care of the necessary variable bindings.

### 6.4.2 Benefits of Abstraction through Variables and Unification

Reconsider our example domain II of UNIX command sequences.

**Example 6.3** UNIX users often reuse a newly created directory in subsequent commands such as in $\mathtt{mkdir(vt100x)}, \mathtt{cd(vt100x)}, \mathtt{ls(vt100x)}$. ○

Unification should allow us to elegantly employ this information because it allows us to specify that, after observing the created directory, the model makes a transition

into a state where the newly created directory is used:

$$p_1 : \texttt{cd}(\texttt{Dir}, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, \texttt{com})$$
$$\text{and } p_2 : \texttt{cd}(\_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, \texttt{com})$$

If the first transition is followed, the `cd` command will move to the newly created directory; if the second transition is followed, it is not specified, which directory `cd` will move to. Thus, the logical hidden Markov model captures the reuse of created directories as an argument of future commands. Moreover, the logical hidden Markov model encodes the simplest possible case to show the benefits of unification. At any time, we know exactly the state we are in, and functors are not used. Therefore, we left out the abstract output symbols associated with abstract transitions. In total, the logical hidden Markov model $U$, modeling the reuse of directories, consists of 542 parameters only but still covers more than 451000 (ground) states. The complete model can be found in the Appendix A.1.

To empirically investigate the benefits of unification, we compare $U$ with the variant $N$ of $U$ where no variables are shared, i.e., no unification is used such that for instance the first transition above is not allowed, see Appendix A.1. $N$ has 164 parameters less than $U$. We computed the following zero-one win function

$$f(\mathsf{O}) = \begin{cases} 1 & \text{if } \left[\log P_U(\mathsf{O}) - \log P_N(\mathsf{O})\right] > 0 \\ 0 & \text{otherwise} \end{cases}$$

leave-one-out cross-validated on UNIX shell logs collected by Greenberg [1988]. Overall, the data consists of 168 users of four groups: computer scientists, nonprogrammers, novices and others. About 300000 commands have been logged with an average of 110 sessions per user. We present here results for a subset of the data. We considered all computer scientist sessions in which at least a single `mkdir` command appears. These yield 283 logical sequences over in total 3286 ground atoms. The LOO win was 81.63%. Other LOO statistics are also in favor of $U$:

| | training | | test | |
|---|---|---|---|---|
| | $\log P(\mathbf{O})$ | $\log \frac{P_U(\mathbf{O})}{P_N(\mathbf{O})}$ | $\log P(\mathsf{O})$ | $\log \frac{P_U(\mathsf{O})}{P_N(\mathsf{O})}$ |
| $U$ | $-11361.0$ | 1795.3 | $-42.8$ | 7.91 |
| $N$ | $-13157.0$ | | $-50.7$ | |

Thus, although $U$ has 164 parameters more than $N$, it shows a better generalization performance. A pattern often found in $U$ was [24]

$$0.15 : \texttt{cd}(\texttt{Dir}, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, \texttt{com})$$
$$\text{and } 0.08 : \texttt{cd}(\_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, \texttt{com})$$

favoring changing to the directory just made. This cannot be captured in $N$

$$0.25 : \texttt{cd}(\_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, \texttt{com}).$$

[24] The sum of probabilities is not the same ($0.15 + 0.08 = 0.23 \neq 0.25$) because of the use of pseudo counts and because of the subliminal non-determinism (w.r.t. abstract states) in $U$, i.e., in case that the first transition fires, the second one also fires.

The results clearly show that abstraction through variables and unification can be beneficial for some applications, i.e., **(B1)** and **(B2)** hold.

## 6.5 Real World Applications

Our intentions here are to investigate whether logical hidden Markov models can be applied to real world domains. More precisely, we will investigate whether benefits **(B1)** and **(B2)** can also be exploited in real world application domains. Additionally, we will investigate whether

**(B3)** logical hidden Markov models are competitive with ILP algorithms that can also utilize unification and abstraction through variables, and
**(B4)** logical hidden Markov models can handle tree-structured data similar to PCFGs.

To this aim, we conducted experiments on two bioinformatics application domains: protein fold recognition [Kersting et al., 2003b] and mRNA signal structure detection [Horváth et al., 2001]. Both application domains are multiclass problems with five different classes each.

### 6.5.1 Methodology

In order to tackle the multiclass problem with logical hidden Markov models, we followed a *plug-in estimate* approach. Let $\{c_1, c_2, \ldots, c_k\}$ be the set of possible classes. Given a finite set of training examples $\{(x_i, y_i)\}_{i=1}^n \subseteq \mathcal{X} \times \{c_1, c_2, \ldots, c_n\}$, one tries to find $f : \mathcal{X} \to \{c_1, c_2, \ldots, c_k\}$

$$f(x) = \arg \max_{c \in \{c_1, c_2, \ldots, c_k\}} P(x \mid M, \boldsymbol{\lambda}_c^*) \cdot P(c) \; . \tag{6.3}$$

with low approximation error on the training data as well as on unseen examples. In Equation (6.3), $M$ denotes the model structure, which is the same for all classes, $\boldsymbol{\lambda}_c^*$ denotes the maximum likelihood parameters of $M$ for class $c$ estimated on the training examples with $y_i = c$ only, and $P(c)$ is the prior class distribution.

We implemented the Baum-Welch algorithm (with pseudocounts $m$, see line 3) for maximum likelihood parameter estimation using the Prolog system Yap-4.4.4. In all experiments, we set $m = 1$ and let the Baum-Welch algorithm stop if the change in log-likelihood was less than 0.1 from one iteration to the next. The experiments were ran on a Pentium-IV 3.2 GHz Linux machine.

### 6.5.2 Protein Fold Recognition

Protein fold recognition is concerned with how proteins fold in nature, i.e., their three-dimensional structures. This is an important problem as the biological functions of proteins depend on the way they fold. A common approach is to use database searches to find proteins (of known fold) similar to a newly discovered protein (of unknown fold). To facilitate protein fold recognition, several expert-based classification schemes

of proteins have been developed that group the current set of known protein structures according to the similarity of their folds. For instance, the *structural classification of proteins* [Hubbard et al., 1997] (SCOP) database hierarchically organizes proteins according to their structures and evolutionary origin. From a machine learning perspective, SCOP induces a classification problem: given a protein of unknown fold, assign it to the best matching group of the classification scheme. This *protein fold classification* problem has been investigated by Turcotte et al. [2001] based on the inductive logic programming (ILP) system PROGOL and by Kersting et al. [2003b] based on logical hidden Markov models.

The secondary structure of protein domains[25] can elegantly be represented as logical sequences.

**Example 6.4** The secondary structure of the Ribosomal protein L4 is represented as

$$\texttt{st}(\texttt{null},2),\texttt{he}(\texttt{right},\texttt{alpha},6),\texttt{st}(\texttt{plus},2),\texttt{he}(\texttt{right},\texttt{alpha},4),$$
$$\texttt{st}(\texttt{plus},2),\texttt{he}(\texttt{right},\texttt{alpha},4),\texttt{st}(\texttt{plus},3),\texttt{he}(\texttt{right},\texttt{alpha},4),$$
$$\texttt{st}(\texttt{plus},1),\texttt{he}(\texttt{hright},\texttt{alpha},6)$$

Here, helices of a certain type, orientation and length $\texttt{he}(\mathit{HelixType}, \mathit{HelixOrientation}, \mathit{Length})$, and strands of a certain orientation and length $\texttt{st}(\mathit{StrandOrientation}, \mathit{Length})$ are atoms over logical predicates. ∘

The application of traditional HMMs to such sequences requires one to either ignore the structure of helices and strands, which results in a loss of information, or to take all possible combinations (of arguments such as orientation and length) into account, which leads to a combinatorial explosion in the number of parameters

The results reported by Kersting et al. [2003b] indicate that logical hidden Markov models are well-suited for protein fold classification: the number of parameters of a logical hidden Markov model can by an order of magnitude be smaller than the number of a corresponding HMM (120 versus approximately 62000) and the generalization performance, a 74% accuracy, is comparable to Turcotte et al.'s [2001] result based on the ILP system Progol, a 75% accuracy. Kersting et al., however, do not cross-validate their results nor investigate — as it is common in bioinformatics — the impact of primary sequence similarity on the classification accuracy. For instance, the two most commonly requested ASTRAL subsets are the subset of sequences with less than 95% identity to each other (95 cut) and with less than 40% identity to each other (40 cut). Motivated by this, we conducted the following new experiments.

The data consists of logical sequences of the secondary structure of protein domains. As in [Kersting et al., 2003b], the task is to predict one of the five most populated SCOP folds of alpha and beta proteins (a/b): TIM beta/alpha-barrel (fold 1), NAD(P)-binding Rossmann-fold domains (fold 2), Ribosomal protein L4 (fold 23), Cysteine hydrolase (fold 37), and Phosphotyrosine protein phosphatases I-like (fold 55). The class of a/b proteins consists of proteins with mainly parallel beta sheets (beta-alpha-beta units). The data have been extracted automatically from the ASTRAL dataset version 1.65 [Chandonia et al., 2004] for the 95 cut and for the 40 cut.

[25] A domain can be viewed as a sub-section of a protein, which appears in a number of distantly related proteins and which can fold independently of the rest of the protein.

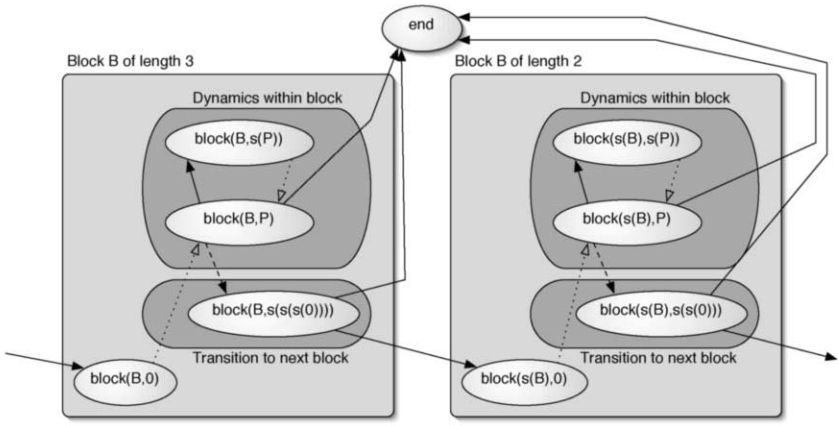



**Figure 6.2.** Scheme of a left-to-right *logical hidden Markov* block *model*.

As in [Kersting et al., 2003b], we consider strands and helices only, i.e., coils and isolated strands are discarded. For the 95 cut, this yields 816 logical sequences consisting of in total 22210 ground atoms. The number of sequences in the classes are listed as 293, 151, 87, 195, and 90. For the 40 cut, this yields 523 logical sequences consisting of in total 14986 ground atoms. The number of sequences in the classes are listed as 182, 100, 66, 122, and 53.

**Logical Hidden Markov Model Structure:** The used logical hidden Markov model structure follows a left-to-right block topology, see Figure 6.2, to model blocks of consecutive helices (resp. strands). Being in a *Block* of some size $s$, say 3, the model will remain in the same block for $s = 3$ time steps. A similar idea has been used by Koivisto et al. [2002, 2004] to model haplotypes. In contrast to common HMM block models [Won et al., 2004], the transition parameters are shared within each block and one can ensure that the model makes a transition to the next state $\mathbf{s}(Block)$ only at the end of a block; in our example after exactly 3 intra-block transitions. Furthermore, there are specific abstract transitions for all helix types and strand orientations to model the priori distribution, the intra- and the inter-block transitions. The number of blocks and their sizes were chosen according to the empirical distribution over sequence lengths in the data so that the model fits well the dynamics at the beginning and the end of protein domains. This yields the following block structure


where the numbers denote the positions within protein domains. Furthermore, note that the last block gathers all remaining transitions. The blocks themselves are mod-

elled using hidden abstract states over

$$\texttt{hc}(\mathit{HelixType}, \mathit{HelixOrientation}, \mathit{Length}, \mathit{Block})$$
$$\text{and } \texttt{sc}(\mathit{StrandOrientation}, \mathit{Length}, \mathit{Block}) \ .$$

Here, *Length* denotes the number of consecutive bases the structure element consists of. The length was discretized into 10 bins such that the original lengths were uniformally distributed. In total, the logical hidden Markov model has 295 parameters. The corresponding HMM without parameter sharing has more than 65200 parameters. This clearly confirms **(B1)**.

**Results:** We performed a 10-fold cross-validation. On the 95 cut dataset, the accuracy was 76% and took approximately 25 minutes per cross-validation iteration; on the 40 cut, the accuracy was 73% and took approximately 12 minutes per cross-validation iteration. The results validate Kersting et al.'s [2003b] results and, in turn, clearly show that **(B3)** holds. Moreover, the novel results on the 40 cut dataset indicate that the similarities detected by the logical hidden Markov models between the protein domain structures were not accompanied by high sequence similarity.

### 6.5.3 mRNA Signal Structure Detection

mRNA sequences consist of bases (guanine, adenine, uracil, cytosine) and fold intramolecularly to form a number of short base-paired stems [Durbin et al., 1998]. This base-paired structure is called the *secondary structure* of the mRNA, cf. Figures 6.3 and 6.4. The secondary structure contains special subsequences called signal structures that are responsible for special biological functions, such as RNA-protein interactions and cellular transport. The function of each signal structure class is based on the common characteristic binding site of all class elements. The elements are not necessarily identical but very similar. They can vary in topology (tree structure), in size (number of constituting bases) and in base sequence.

The goal of our experiments was to recognize instances of signal structures classes in mRNA molecules. The first application of relational learning to recognize the signal structure class of mRNA molecules was described in [Bohnebeck et al., 1998, Horváth et al., 2001] where the relational instance-based learner RIBL was applied. The dataset [26] we used was similar to the one described by Horváth et al.. It was composed of 15 and 5 SECIS (Selenocysteine Insertion Sequence), 27 IRE (Iron Responsive Element), 36 TAR (Trans Activating Region) and 10 histone stemloops constituting five classes.

The secondary structure is composed of different building blocks such as stacking region, hairpin loops, interior loops etc. In contrast to the secondary structure of proteins that forms chains, the secondary structure of mRNAs forms a tree. As trees

[26] The dataset is not the same as described in [Horváth et al., 2001] because we could not obtain the original dataset. We will compare to the smaller data set used in [Horváth et al., 2001], which consisted of 66 signal structures and is very close to our data set. On a larger data set (with 400 structures) Horváth et al. report an error rate of 3.8% .

can not easily be handled using HMMs, mRNA secondary structure data is more challenging than that of proteins. Moreover, Horváth et al. [2001] report that making the tree structure available to RIBL as background knowledge had an influence on the classification accuracy. More precisely, using a simple chain representation RIBL achieved a 77.2% leave-one-out cross-validation (LOO) accuracy whereas using the tree structure as background knowledge RIBL achieved a 95.4% LOO accuracy.

We followed Horváth et al.'s experimental setup, that is, we adapted their data representations to logical hidden Markov models and compared a chain model with a tree model.

**Chain Representation:** In the *chain* representation (see also Figure 6.3), signal structures are described by

$$\begin{aligned} &\texttt{single}(\mathit{TypeSingle}, \mathit{Position}, \mathit{Acid}) \\ \text{or } &\texttt{helical}(\mathit{TypeHelical}, \mathit{Position}, \mathit{Acid}, \mathit{Acid}) \ . \end{aligned}$$

Depending on its type, a structure element is represented by either `single`/3 or `helical`/4. Their first argument *TypeSingle* (resp. *TypeHelical*) specifies the type of the structure element, i.e., `single`, `bulge3`, `bulge5`, `hairpin` (resp. `stem`). The argument *Position* is the position of the sequence element within the corresponding structure element counted down, i.e.[27], $\{\mathtt{n}^{13}(\mathtt{0}), \mathtt{n}^{12}(\mathtt{0}), \ldots, \mathtt{n}^{1}(\mathtt{0})\}$. The maximal position was set to 13 as this was the maximal position observed in the data. The last argument encodes the observed nucleotide (pair).

The used logical hidden Markov model structure follows again the left-to-right block structure shown in Figure 6.2. Its underlying idea is to model blocks of consecutive helical structure elements. The hidden states are modelled using

$$\begin{aligned} &\texttt{single}(\mathit{TypeSingle}, \mathit{Position}, \mathit{Acid}, \mathit{Block}) \\ \text{and } &\texttt{helical}(\mathit{TypeHelical}, \mathit{Position}, \mathit{Acid}, \mathit{Acid}, \mathit{Block}) \ . \end{aligned}$$

Being in a *Block* of consecutive helical (resp. single) structure elements, the model will remain in the *Block* or transition to a `single` element. The transition to a single (resp. helical) element only occurs at *Position* $\mathtt{n}(\mathtt{0})$. At all other positions $\mathtt{n}(\mathit{Position})$, there were transitions from helical (resp. single) structure elements to helical (resp. single) structure elements at *Position* capturing the dynamics of the nucleotide pairs (resp. nucleotides) within structure elements. For instance, the transitions for block $\mathtt{n}(\mathtt{0})$ at position $\mathtt{n}(\mathtt{n}(\mathtt{0}))$ were

$$\begin{aligned}
a &: \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{0}), \mathtt{X}, \mathtt{Y}, \mathtt{n}(\mathtt{0})) \xleftarrow{p_a:\mathtt{he}(\mathtt{stem},\mathtt{n}(\mathtt{0}),\mathtt{X},\mathtt{Y})} \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{n}(\mathtt{0})), \mathtt{X}, \mathtt{Y}, \mathtt{n}(\mathtt{0})) \\
b &: \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{0}), \mathtt{Y}, \mathtt{X}, \mathtt{n}(\mathtt{0})) \xleftarrow{p_b:\mathtt{he}(\mathtt{stem},\mathtt{n}(\mathtt{0}),\mathtt{X},\mathtt{Y})} \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{n}(\mathtt{0})), \mathtt{X}, \mathtt{Y}, \mathtt{n}(\mathtt{0})) \\
c &: \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{0}), \mathtt{X}, _, \mathtt{n}(\mathtt{0})) \xleftarrow{p_c:\mathtt{he}(\mathtt{stem},\mathtt{n}(\mathtt{0}),\mathtt{X},\mathtt{Y})} \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{n}(\mathtt{0})), \mathtt{X}, \mathtt{Y}, \mathtt{n}(\mathtt{0})) \\
d &: \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{0}), _, \mathtt{Y}, \mathtt{n}(\mathtt{0})) \xleftarrow{p_d:\mathtt{he}(\mathtt{stem},\mathtt{n}(\mathtt{0}),\mathtt{X},\mathtt{Y})} \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{n}(\mathtt{0})), \mathtt{X}, \mathtt{Y}, \mathtt{n}(\mathtt{0})) \\
e &: \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{0}), _, _, \mathtt{n}(\mathtt{0})) \xleftarrow{p_e:\mathtt{he}(\mathtt{stem},\mathtt{n}(\mathtt{0}),\mathtt{X},\mathtt{Y})} \mathtt{he}(\mathtt{stem}, \mathtt{n}(\mathtt{n}(\mathtt{0})), \mathtt{X}, \mathtt{Y}, \mathtt{n}(\mathtt{0}))
\end{aligned}$$

[27] $\mathtt{n}^m(\mathtt{0})$ is shorthand for the recursive application of the functor `n` on `0` $m$ times, i.e., for position $m$.

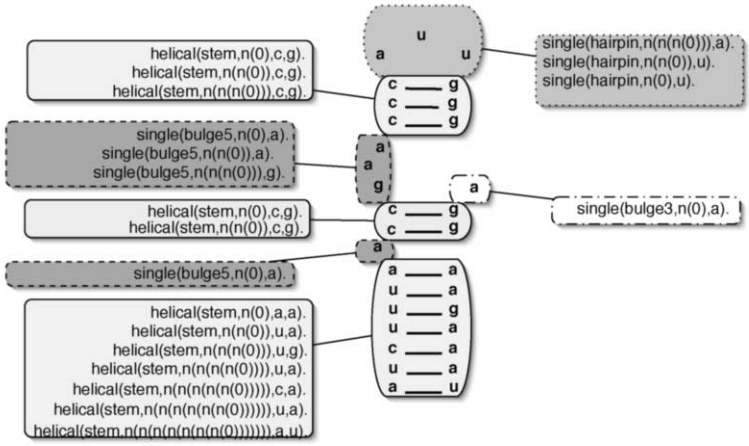



**Figure 6.3.** The *chain* representation of a SECIS signal structure. The ground atoms are ordered clockwise starting with $\mathtt{helical(stem,n(n(n(n(n(n(n(0)))))))},\mathtt{a},\mathtt{u})$ at the lower left-hand side corner.

In total, there were 5 possible blocks as this was the maximal number of blocks of consecutive helical structure elements observed in the data. Overall, the logical hidden Markov model has 702 parameters. In contrast, the corresponding HMM has more than 16600 transitions validating **(B1)**.

**Results:** The LOO test log-likelihood was $-63.7$, and an EM iteration took on average 26 seconds.

Without the unification-based transitions *b*-*d*, i.e., using only the abstract transitions

$$a : \mathtt{he(stem,n(0),X,Y,n(0))} \xleftarrow{p_a:\mathtt{he(stem,n(0),X,Y)}} \mathtt{he(stem,n(n(0)),X,Y,n(0)))}$$
$$e : \mathtt{he(stem,n(0),\_,\_,n(0))} \xleftarrow{p_e:\mathtt{he(stem,n(0),X,Y)}} \mathtt{he(stem,n(n(0)),X,Y,n(0))),}$$

the model has 506 parameters. The LOO test log-likelihood was $-64.21$, and an EM iteration took on average 20 seconds. The difference in LOO test log-likelihood is statistically significant (paired $t$-test, $p = 0.01$).

Omitting even transition $a$, the LOO test log-likelihood dropped to $-66.06$, and the average time per EM iteration was 18 seconds. The model has 341 parameters. The difference in average LOO log-likelihood is statistically significant (paired $t$-test, $p = 0.001$).

The results clearly show that unification can yield better LOO test log-likelihoods, i.e., **(B2)** holds.

**Tree Representation:** In the *tree* representation (see Figure 6.4 **(a)**), the idea is to capture the tree structure formed by the secondary structure elements, see Fig-

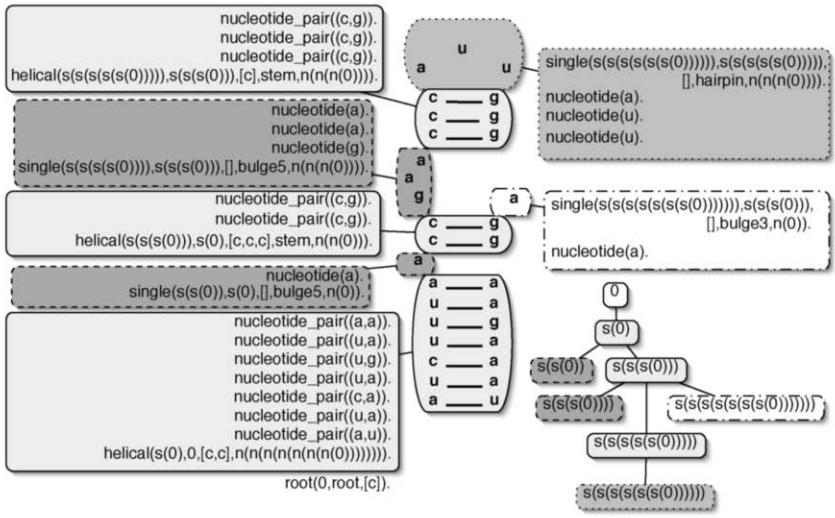



**Figure 6.4.** The *tree* representation of a SECIS signal structure.**(a)** The logical sequence, i.e., the sequence of ground atoms representing the SECIS signal structure. The ground atoms are ordered clockwise starting with $\texttt{root}(0, \texttt{root}, [\texttt{c}])$ in the lower left-hand side corner. **(b)** The tree formed by the secondary structure elements.

ure 6.4 **(b)**. Each training instance is described as a sequence of ground facts over

$$\texttt{root}(0, \texttt{root}, \#\mathit{Children}),$$
$$\texttt{helical}(\mathit{ID}, \mathit{ParentID}, \#\mathit{Children}, \mathit{Type}, \mathit{Size}),$$
$$\texttt{nucleotide\_pair}(\mathit{BasePair}),$$
$$\texttt{single}(\mathit{ID}, \mathit{ParentID}, \#\mathit{Children}, \mathit{Type}, \mathit{Size}),$$
$$\texttt{nucleotide}(\mathit{Base})\ .$$

Here, *ID* and *ParentID* are natural numbers $0, \texttt{s}(0), \texttt{s}(\texttt{s}(0)), \ldots$ encoding the child-parent relation, $\#\mathit{Children}$ denotes the number of children $[], [\texttt{c}], [\texttt{c}, \texttt{c}], \ldots$, *Type* is the type of the structure element such as $\texttt{stem}, \texttt{hairpin}, \ldots$, and *Size* is a natural number $0, \texttt{n}(0), \texttt{n}(\texttt{n}(0)), \ldots$ Atoms $\texttt{root}(0, \texttt{root}, \#\mathit{Children})$ are used to root the topology. The maximal $\#\mathit{Children}$ was 9 and the maximal *Size* was 13 as this was the maximal value observed in the data.

As trees can not easily be handled using HMMs, we used a logical hidden Markov model, which basically encodes a PCFG. Due to Theorem 6.1, this is possible. The used logical hidden Markov model structure can be found in Appendix A.2. It processes the mRNA trees in in-order. Unification is only used for parsing the tree. As for the chain representation, we used a *Position* argument in the hidden states to encode the dynamics of nucleotides (nucleotide pairs) within secondary structure elements. The maximal *Position* was again 13. In contrast to the chain representation,

nucleotide pairs such as $(\mathtt{a}, \mathtt{u})$ are treated as constants. Thus, the argument *BasePair* consists of 16 elements.

**Results:** The LOO test log-likelihood was $-55.56$. Thus, exploiting the tree structure yields better probabilistic models. On average, an EM iteration took 14 seconds. Overall, the result shows that **(B4)** holds.

Although the Baum-Welch algorithm attempts to maximize a different objective function, namely the likelihood of the data, it is interesting to compare logical hidden Markov models and RIBL in terms of classification accuracy.

**Classification Accuracy:** On the *chain* representation, the LOO accuracies of all logical hidden Markov models were 99% (92/93). This is a considerable improvement on RIBL's 77.2% (51/66) LOO accuracy for this representation. On the *tree* representation, the logical hidden Markov model also achieved a LOO accuracy of 99% (92/93). This is comparable to RIBL's LOO accuracy of 97% (64/66) on this kind of representation.

Thus, already the chain logical hidden Markov models show marked increases in LOO accuracy when compared to RIBL [Horváth et al., 2001]. In order to achieve similar LOO accuracies, Horváth et al. had to make the tree structure available to RIBL as background knowledge. For logical hidden Markov models, this had a significant influence on the LOO test log-likelihood, but not on the LOO accuracies. This clearly supports **(B3)**. Moreover, according to Horváth et al., the mRNA application can also be considered a success in terms of the application domain, although this was not the primary goal of our experiments. There exist also alternative parameter estimation techniques and other models, such as covariance models [Eddy and Durbin, 1994] or pair hidden Markov models [Sakakibara, 2003], which might have been used as well as a basis for comparison. However, as logical hidden Markov models employ inductive logic programming principles, it is appropriate to compare with other systems within this paradigm such as RIBL.

§ 7

# Learning the Structure of Logical HMMs *

*... in which the structure learning problem for logical hidden Markov models is formalized and a structural, generalized Expectation-Maximization approach for solving the problem is presented and experimentally evaluated ...*

The compactness and comprehensibility of logical hidden Markov models comes at the expense of a more complex model selection respectively structure learning problem. Structure selection for logical hidden Markov models is a significant problem for many

* Builds on [Kersting et al., 2003a, Kersting and Raiko, 2005].

reasons. First, eliciting logical hidden Markov models from experts can be a laborious and expensive process. Second, HMMs are commonly learned by estimating the maximum likelihood parameters of a fixed, fully connected model. Such an approach is not feasible for logical hidden Markov models as different abstraction levels have to be explored. Third, logical hidden Markov models are strictly more expressive than HMMs as shown in Theorem 6.1 and in the experiments on tree-structured mRNA data in Section 6.5.3. Finally, parameter estimation for logical hidden Markov models is a costly nonlinear optimization problem, so the naïve search is infeasible.

We propose SAGEM [28] for selecting the structure of logical hidden Markov models from data. SAGEM adapts Friedman's *structural EM* [1997]. It combines a *generalized expectation maximization* (GEM) algorithm, which optimizes parameters, with structure search for model selection using ILP refinement operators. Thus, SAGEM explores different abstraction levels due to ILP refinement operators, and, due to a GEM approach, it reduces the selection problem to a more efficiently solvable one.

## 7.1 The Learning Setting: Probabilistic Learning from Proofs

For traditional HMMs, the learning problem basically collapses to parameter estimation (i.e., estimating the transition probabilities) because HMMs can be considered to be fully connected. For logical hidden Markov models, however, we have to account for different abstraction levels. We treat the model selection problem as an instance of the probabilistic ILP learning problem, see Definition 2.26, for learning from *possible* traces *only*. More precisely:

**Definition 7.1** (Learning Problem) **Given** a set $\mathbf{O} = \{\mathsf{O}_1, \ldots, \mathsf{O}_m\}$ of possible examples independently sampled from the same distribution, a set $\mathcal{M}$ of logical hidden Markov models subject to some language bias $\mathcal{L}$, the distribution $P(\mathsf{O} \mid M)$ for each $M \in \mathcal{M}$ as specified in Section 6 as probabilistic covers relation, and a scoring function $score_{\mathbf{O}} : \mathcal{M} \mapsto \mathbb{R}$, **find** a hypothesis $M^* \in \mathcal{M}$ that maximizes $score_{\mathbf{O}}$. ∘

Each example, i.e., *data case* $O_i \in \mathbf{O}$ is a sequence $O_i = \mathtt{o}_{\mathtt{i},1}\mathtt{o}_{\mathtt{i},2} \ldots \mathtt{o}_{\mathtt{i},\mathtt{T}_\mathtt{i}}$ of ground atoms and describes the observations evolving over time. For the sake of simplicity, we assume that $T_1 = T_2 = \ldots = T_m$.

**Example 7.2** In the user modeling domain a data case could be $\mathtt{emacs(lohmms)}, \mathtt{ls}, \mathtt{emacs(lohmms)}$ . ∘

The corresponding evolution of the system's state over time $H_i = \mathtt{h}_{\mathtt{i},0}\mathtt{h}_{\mathtt{i},1} \ldots \mathtt{h}_{\mathtt{i},\mathtt{T}_\mathtt{i}+1}$ is hidden, i.e. , not specified in $O_i$.

**Example 7.3** Continuing our running example, we do not know whether $\mathtt{emacs(lohmms)}$ has been generated by $\mathtt{emacs(lohmms, other)}$ or $\mathtt{emacs(lohmms, tex)}$ . ∘

The *hypothesis space* $\mathcal{M}$ consists of all candidate logical hidden Markov models to be considered during search. We assume the alphabet $\Sigma$ of the logical HMMs in $\mathcal{M}$ to

[28] German for 'say EM'

**Algorithm II.4**: Naïve approach for learning the structure of logical hidden Markov models from data.

```
1 k := 0
2 let M^k be some proper initial model structure
3 let λ^k = arg max_λ score_O(M^k, λ)
4 repeat
5 |   find a model structure M^{k+1} ∈ ρ(M^k) maximizing score_O(M^{k+1}, λ)
6 |   let λ^{k+1} = arg max_λ score_O(M^{k+1}, λ)
7 |   k := k + 1
8 until score_O(M^{k-1}, λ^{k-1}) ≥ score_O(M^k, λ^k), i.e., no improvement in score
```

be given. Thus, the possible constants, which can be selected by $\mu$ are apriori known. Each $M \in \mathcal{M}$ is parameterized by a vector $\boldsymbol{\lambda}_M$. Each (legal) choice of $\boldsymbol{\lambda}_M$ defines a probability distribution $P(\cdot \mid M, \boldsymbol{\lambda}_M)$ over $\bigotimes_t \mathrm{hb}(\Sigma)$, ie., over sequences over $\mathrm{hb}(\Sigma)$. For the sake of simplicity, we will denote the underlying logic program (i.e., the set of abstract transitions without associated probability values) by $M$ and abbreviate $\boldsymbol{\lambda}_M$ by $\boldsymbol{\lambda}$ as long as no ambiguities can arise. Furthermore, the syntactic bias $\mathcal{L}$ on the transitions to be induced is a parameter of our framework, as usual in ILP Nédellec et al. [1996]. For instance in the experiments, we only consider transitions that obey the type constraints induced by the predicates.

As *score*, we employ a penalized log-likelihood

$$score_{\mathbf{O}}(M, \boldsymbol{\lambda}) = \log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) - Pen(M, \boldsymbol{\lambda}, \mathbf{O}) \; . \tag{7.1}$$

Here, $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda})$ is the *log-likelihood* of the current of model $(M, \boldsymbol{\lambda})$. The higher the log-likelihood is, the closer $(M, \boldsymbol{\lambda})$ models the probability distribution induced by the data. The second term, $Pen(M, \boldsymbol{\lambda}, \mathbf{O})$, is a penalty function that biases the scoring function to prefer simpler models. Motivated by the *minimum description length* score for Bayesian networks [Lam and Bacchus, 1994], we use the simple penalty

$$Pen(M, \boldsymbol{\lambda}, \mathbf{O}) = |\Delta| \log(m)/2 \tag{7.2}$$

where $m$ is the number of training examples, cf. Definition 7.1, and $|\Delta|$ is the number of abstract transitions in $M$, cf. Definition 5.13. The penalty is independent of the model parameters $\boldsymbol{\lambda}$ and therefore it can be neglected when estimating parameters. We assume that each $M$ covers all possible observation sequences (over the given language $\Sigma$). This guarantees that all new data cases will get a positive likelihood.

## 7.2 A Naïve Learning Algorithm

A simple way of selecting a model structure is the greedy approach described in Algorithm II.4. It takes as input an initial model $M^0$ and the data $\mathbf{O}$. At each stage $k$ it chooses a model structure and parameters among the current best model $M^k$ and its neighbours $\rho(M^k)$ (see below) that have the highest score. It stops, when there is

no improvement in score. For scoring models in lines 3 and 5, the model parameters $\boldsymbol{\lambda}$ are randomly initialized.

We will now show how to traverse the hypotheses space and how to estimate parameters for a hypothesis in order to score it. That is, we will make line 5 more concrete.

### 7.2.1 Traversing the Hypotheses Space

An obvious candidate for the initial hypothesis $M^0$ (,which we also used in our experiments) is the fully connected logical hidden Markov model built over all maximally general atoms over $\Sigma$, i.e., expressions of the form $\mathtt{r}(\mathtt{X}_1, ..., \mathtt{X}_\mathtt{m})$, where the $\mathtt{X}_\mathtt{i}$ are different variables.

Now, to traverse the hypothesis space $\mathcal{M}$, we have to compute all neighbours of the currently best hypothesis $M^k$. To do so, we adapt the refinement operators traditionally used in ILP, cf. Definition 2.13. More precisely, for the language bias $\mathcal{L}$ used in the experiments in Section 7.4, we used the refinement operator $\rho : \mathcal{M} \mapsto 2^{\mathcal{M}}$ that selects a single clause $\mathtt{T} \equiv p : \mathtt{H} \overset{\mathtt{0}}{\leftarrow} \mathtt{B} \in \mathcal{M}$ and adds a minimal specialization $\mathtt{T}' \equiv p : \mathtt{H}' \overset{\mathtt{0}'}{\leftarrow} \mathtt{B}'$ of $\mathtt{T}$ to $\mathcal{M}$ (with respect to $\theta$-subsumption). Specializing a single abstract transition means instantiating or unifying variables, i.e., $\mathtt{T}' \equiv \mathtt{T}\theta$ for some substitution $\theta$. In contrast to ILP, however, we have to be a little bit more careful when adding $\mathtt{T}'$ to $M^k$. We have to ensure that

(1) the same observation and hidden state sequences are still covered. For a similar reason, we applied the LGG when learning stochastic logic programs, cf. Section 2.3.2, only to clauses that define the same predicate, that contain the same predicates, and whose (reduced) LGG also has the same length as the original clauses. Furthermore, we have to ensure that
(2) the list of bodies $\mathbf{B}'$ after adding $\mathtt{T}'$ should remain well-founded, that is, for each ground state, there is a unique maximally specific body in $\mathbf{B}'$.

These conditions together guarantee that the most specific body corresponding to a state always exists and is unique. Condition (1) can only be violated if $\mathtt{B}' \notin \mathbf{B}$ [29]. In this case, we add transitions with $\mathtt{B}'$ and maximally general heads and corresponding observations. Now, we compute the maximal specific $\mathtt{B}'' \in \mathbf{B}$ that subsumes $\mathtt{B}'$. If it does not exist or $\mathtt{B}'$ subsumes $\mathtt{B}''$, we stop. Otherwise, we add corresponding transitions for the mgu of $\mathtt{B}'$ and $\mathtt{B}''$. Condition (2) is established analogously. We keep the list of bodies well-founded by adding new bodies (and therefore abstract transitions) in a similar way as described above.

**Example 7.4** Consider refining the logical hidden Markov models in Figure 7.1. When adding $\mathtt{ls}(\mathtt{U}) \overset{\mathtt{latex(lohmm)}}{\longleftarrow} \mathtt{latex}(\mathtt{lohmm}, \mathtt{U})$ , i.e., introducing the more specific abstract state $\mathtt{latex}(\mathtt{lohmm}, \mathtt{U})$, further variants of the same abstract transition but with different heads have to be added. Otherwise condition (1) would be violated as the resulting logical hidden Markov model does not cover the same sequences as

[29] Otherwise, i.e., if $\mathtt{B}' \in \mathbf{B}$, $\mathbf{B}$ was already well-founded and remains well-founded when adding the refined abstract transition.

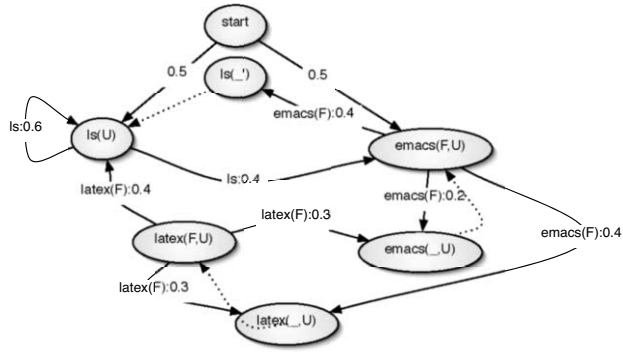



**Figure 7.1.** Another logical hidden Markov model.

the original one; the state `latex(lohmm,U)` can only be left via `ls(U)` and not for instance via `emacs(_,U)`. On the other hand, we have to be careful when subsequently adding abstract transitions for the body `latex(F,tex)`. The problem is that we do not know which abstract body to select in state `latex(lohmm,tex)`. To fulfill condition (2), we need to add abstract transitions for an additional, third abstract state `latex(lohmm,tex)`, too. ○

### 7.2.2 Parameter Estimation

The most common approach for parameter estimation of HMMs in the presence of hidden variables and missing observations is the Baum-Welch algorithm, see Section 6.3, which is an instance of the EM algorithm [Dempster et al., 1977, McLachlan and Krishnan, 1997]. Because SAGEM, which we will introduce in Section 7.3, builds upon the EM algorithm, let us briefly review the basics of the EM algorithm.

In each iteration $l+1$, the EM algorithm performs two steps:

**(E-Step)** Compute the expectation of the log-likelihood given the old model $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$, i.e.,

$$Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) = E\left[\log P(\mathbf{O}, \mathbf{H} \mid M^k, \boldsymbol{\lambda}) \mid M^k, \boldsymbol{\lambda}^{k,l}\right] .$$

For logical HMMs, $\mathbf{O}, \mathbf{H}$ is the completion of $\mathbf{O}$ where $\mathbf{H}$ denotes hidden state sequences, which could have generated $\mathbf{O}$. The current model $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$ give us the conditional distribution governing $\mathbf{H}$, and $E[\cdot|\cdot]$ denotes the expectation over it. The function $Q$ is called the *expected score*.

**(M-Step)** Maximize the expected score $Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$ w.r.t. $\boldsymbol{\lambda}$, i.e.,

$$\boldsymbol{\lambda}^{k,l+1} = \arg\max_{\boldsymbol{\lambda}} Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) . \tag{7.3}$$

**Algorithm II.5**: sagEM: A *structural generalized expectation-maximization* approach for learning the structure of logical hidden Markov models from data.

```
let k = 0
let M^k be some proper initial model structure
initialize λ^{0,0} randomly
repeat
    let l = 0
    repeat
        let λ^{k,l+1} = arg max_λ Q(M^k, λ | M^k, λ^{k,l})
        let l = l + 1
    until convergence, i.e., no improvement in score, or l = l_max
    find model structure M^{k+1} ∈ ρ(M^k) maximizing Q(M^{k+1}, λ | M^k, λ^{k,l})
    let λ^{k+1,0} = arg max_λ Q(M^{k+1}, λ | M^k, λ^{k,l})
    let k = k + 1
until convergence, i.e., no improvement in score
```

The naïve greedy algorithm can easily be instantiated using the Baum-Welch algorithm. The problem, however, is its huge computational costs. To evaluate a single neighbour, the Baum-Welch algorithm has to run for a reasonable number of iterations in order to get reliable ML estimates of $\boldsymbol{\lambda}^{k'}$. Each iteration requires a full logical hidden Markov model inference on all data cases. In total, the running time per neighbour evaluation is at least $\mathcal{O}(\#\textit{EM iterations} \cdot \textit{size of data})$ .

## 7.3 sagEM: A Structural Generalized EM

To reduce the computational costs, sagEM [30] adapts Friedman's *structural EM* (SEM) [1997]. That is, we take our current model $(M^k, \boldsymbol{\lambda}^k)$ and run the Baum-Welch algorithm, i.e., the EM algorithm for a while to get reasonably completed data. We then fix the completed data cases and use them to compute the ML parameters $\boldsymbol{\lambda}^{k'}$ of each neighbour $M^{k'}$. We choose the neighbour with the best improvement of the score as $(M^{k+1}, \boldsymbol{\lambda}^{k+1})$ and iterate. Algorithm II.5 describes the approach more formally.

The hypotheses space is traversed as described earlier in Section 7.2.1, and again we stop if there is no improvement in score. The following theorem shows that even when the model structure changes in between two iterations of the EM algorithm, improving the expected score $Q$ always improves the log-likelihood as well.

**Theorem 7.5** *If* $Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds, then* $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) > \log P(\mathbf{O} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds.* ∘

The proof is an application of the argumentation for the monotonicity of the EM algorithm by [McLachlan and Krishnan, 1997, Section 3.2 ff.]. To apply the algorithm

[30] German for *'say EM'*

to selecting logical hidden Markov models, we will now show how to choose the best neighbour [31] in line 10.

Let $c(\mathtt{b},\mathtt{h},\mathtt{o})$ denotes the number of times the systems proceeds from ground state $\mathtt{b}$ to ground state $\mathtt{h}$ emitting ground observation $\mathtt{o}$. The expected score in line 10 simplifies to

$$\begin{aligned}
&Q(M,\boldsymbol{\lambda}|M^k,\boldsymbol{\lambda}^{k,l})\\
&= E\left[\log P(\mathbf{O},\mathbf{H}|M,\boldsymbol{\lambda})\middle|M^k,\boldsymbol{\lambda}^{k,l}\right]\\
&= E\left[\log \prod\nolimits_t P(\mathtt{h_{t+1}},\mathtt{o_{t+1}}|\mathtt{h_t},M,\boldsymbol{\lambda})\middle|M^k,\boldsymbol{\lambda}^{k,l}\right]\\
&= E\left[\log \prod\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} P(\mathtt{h},\mathtt{o}|\mathtt{b},M,\boldsymbol{\lambda})^{c(\mathtt{b},\mathtt{h},\mathtt{o})}\middle|M^k,\boldsymbol{\lambda}^{k,l}\right]\\
&= E\left[\sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} c(\mathtt{b},\mathtt{h},\mathtt{o})\cdot\log P(\mathtt{h},\mathtt{o}|\mathtt{b},M,\boldsymbol{\lambda})\middle|M^k,\boldsymbol{\lambda}^{k,l}\right]\\
&= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \underbrace{E\left[c(\mathtt{b},\mathtt{h},\mathtt{o})\middle|M^k,\boldsymbol{\lambda}^{k,l}\right]}_{=:ec(\mathtt{b},\mathtt{h},\mathtt{o})}\cdot\log P(\mathtt{h},\mathtt{o}|\mathtt{b},M,\boldsymbol{\lambda})\ . \qquad (7.4)
\end{aligned}$$

The term $ec(\mathtt{b},\mathtt{h},\mathtt{o})$ in (7.4) denotes the expected counts of making a transition from ground state $\mathtt{b}$ to ground state $\mathtt{h}$ emitting ground observation $\mathtt{o}$. The expectation is taken according to $(M^k,\boldsymbol{\lambda}^{k,l})$.

An analytical solution, however, of the M-step in line 10 seems to be difficult. In HMMs, the updated transition probabilities are simply directly proportional to the expected number of times they are used. In logical hidden Markov models, however, there is an ambiguity: multiple abstract transitions (with the same body), can match the same ground transition $(\mathtt{b},\mathtt{h},\mathtt{o})$. Using *ec* as sufficient statistics makes the M step nontrivial. The solution is to improve Equation (7.4) instead of maximizing it. Such an approach is called *generalized EM*, see [McLachlan and Krishnan, 1997]. To do so, we follow a gradient-based optimization technique. We iteratively perform two steps:

(1) Compute the *gradient* $\nabla_{\boldsymbol{\lambda}}$ of (7.4) with respect to the parameters of a logical hidden Markov model, and, then,
(2) take a step in the direction of the gradient to the point $\boldsymbol{\lambda}+\delta\nabla_{\boldsymbol{\lambda}}$ where $\delta$ is the step-size.

For logical hidden Markov models, the gradient with respect to (7.4) consists of partial derivatives with respect to abstract transition probabilities and to selection probabilities.

Assume that $\lambda$ is the transition probability associated with some abstract transition $\mathtt{T}$. Now, the partial derivative of (7.4) with respect to some parameter $\lambda$ is

$$\begin{aligned}
\frac{\partial Q(M,\boldsymbol{\lambda}\mid M^k,\boldsymbol{\lambda}^{k,l})}{\partial\lambda} &= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} ec(\mathtt{b},\mathtt{h},\mathtt{o})\cdot\frac{\partial\log P(\mathtt{h},\mathtt{o}\mid\mathtt{b},M,\boldsymbol{\lambda})}{\partial\lambda}\\
&= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \frac{ec(\mathtt{b},\mathtt{h},\mathtt{o})}{P(\mathtt{h},\mathtt{o}\mid\mathtt{b},M,\boldsymbol{\lambda})}\cdot\frac{\partial P(\mathtt{h},\mathtt{o}\mid\mathtt{b},M,\boldsymbol{\lambda})}{\partial\lambda} \qquad (7.5)
\end{aligned}$$

[31] For the sake of simplicity, we will not explicitly check that a transition is *maximally specific* for ground states.

The partial derivative of $P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})$ with respect to $\lambda$ can be computed as follows:

$$\begin{aligned}\frac{P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial\lambda} &= \frac{\partial}{\partial\lambda} \sum\nolimits_{\mathtt{T}} P(\mathtt{T} \mid M, \boldsymbol{\lambda}) \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathtt{T})\theta_{\mathtt{H}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathtt{T})\theta_{\mathtt{H}}\theta_{\mathtt{O}}, M) \\ &= \mu(\mathtt{h} \mid \mathrm{head}(\mathtt{T})\theta_{\mathtt{H}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathtt{T})\theta_{\mathtt{H}}\theta_{\mathtt{O}}, M) \qquad (7.6)\end{aligned}$$

Substituting (7.6) back into (7.5) yields

$$\begin{aligned}&\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial\lambda} \\ &\quad= \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \left( \frac{ec(\mathtt{b}, \mathtt{h}, \mathtt{o})}{P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathtt{T})\theta_{\mathtt{H}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathtt{T})\theta_{\mathtt{H}}\theta_{\mathtt{O}}, M) \right) .\end{aligned}$$

The selection probability follows a naïve Bayes approach. Therefore, one can show in a similar way as for transition probabilities that

$$\begin{aligned}\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial\lambda} = \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \Big( &\frac{ec(\mathtt{b}, \mathtt{h}, \mathtt{o})}{P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \sum_{\mathtt{T}} c(\lambda, \mathtt{T}, \mathtt{b}, \mathtt{h}, \mathtt{o}) \cdot \\ &\cdot P(\mathtt{T} \mid M, \boldsymbol{\lambda}) \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathtt{T})\theta_{\mathtt{H}}, M) \cdot \\ &\cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathtt{T})\theta_{\mathtt{H}}\theta_{\mathtt{O}}, M) \Big)\end{aligned}$$

where $c(\lambda, \mathtt{T}, \mathtt{b}, \mathtt{h}, \mathtt{o})$ is the number of times that the domain element associated with $\lambda$ is selected to ground $\mathtt{T}$ with respect to $\mathtt{h}$ and $\mathtt{o}$.

The described method has to be modified to take into account that $\lambda \in [0, 1]$ and that corresponding $\lambda$'s sum to 1.0. For our experiments, we simply follow the approach taken for Bayesian logic programs, see Section 4.3.2, page 67, and reparameterized the problem using

$$\lambda_{ij} = \frac{\beta_{ij})}{(\sum_l \exp(\beta_{il}))} .$$

This enforces the constraints given above, and a local maximum with respect to $\boldsymbol{\beta}$ is also a local maximum with respect to $\boldsymbol{\lambda}$, and vice versa. The gradient with respect to the $\beta_{ij}$'s can be found by computing the gradient with respect to the $\lambda_{ij}$'s and then deriving the gradient with respect to $\boldsymbol{\beta}$ using the chain rule. The derivation follows the same steps as for Bayesian logic programs.

### 7.3.1 Discussion

What are the benefits of sagEM over the naïve approach? The expected ground counts $ec(\mathtt{b}, \mathtt{h}, \mathtt{o})$ are used as the sufficient statistics to evaluate all the neighbours. Evaluating neighbours is thus now independent of the number and length of the data cases, which is an important feature for scaling up. More precisely, the running time

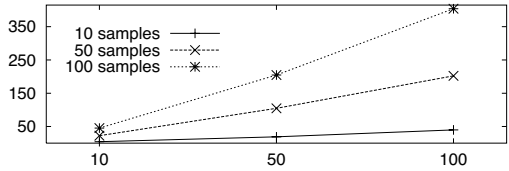



**Figure 7.2.** sagEM's speed-up (y axis), i.e., the ratio of time per EM iteration (in sec.) and time per sagEM's gradient approach for evaluating neighbours. The speed-up is shown for different sequence lengths (x axis) and for different numbers of data cases (curves).

per neighbour evaluation is basically $\mathcal{O}(\#\textit{Gradient iterations} \cdot \#\textit{Ground transitions})$ because sagEM's gradient approach does not perform logical hidden Markov model inference.

The greedy approach does not always suffice. For instance, if two hidden states are equivalent, to make them effectively differ from each other, one needs to make them differ both in visiting probabilities of the state and in behavior in the state, possibly requiring two steps for any positive effect. Fixing the expected counts in sagEM worsens the problem, since changes in visiting probabilities of states do not show up before a logical hidden Markov model inference is made. To overcome this, different search strategies, such as beam search, can be used: Instead of a current hypothesis, a fixed-size set of current hypotheses is considered, and their common neighbourhood is searched for the next set.

To summarize, sagEM explores different abstraction levels due to ILP refinement operators, and, due to a GEM approach, it reduces the neighbourhood evaluation problem to one that is more efficiently solvable.

## 7.4 Experimental Evaluation

We investigate whether sagEM can be applied to real world domains. More precisely, we are interested whether sagEM

**H1** speeds-up neighbour evaluation considerably (compared to the naïve approach);
**H2** finds a comprehensible model;
**H3** works in the presence of transition ambiguity;
**H4** can be applied to real-world domains and is competitive with standard machine learning algorithms such as nearest-neighbour and decision-tree learners.

To this aim, we implemented sagEM using the Prolog system YAP-4.4.4. The experiments were run on a Pentium-III-2.3 GHz machine. For the improvement of the expected score, we adapted the scaled conjugate gradient as implemented in Bishop and Nabney's Netlab library (`http://www.ncrg.aston.ac.uk/netlab/`) with a maximum number of 10 iterations and 5 random restarts.

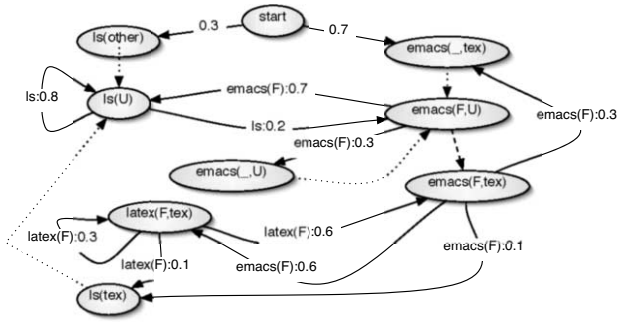



**Figure 7.3.** The original hypothesis for the experiments with synthetic data.

### 7.4.1 Experiments with Synthetic Data

We sampled independently $10, 50, 100$ sequences of length $10, 50, 100$ (100 to 10000 ground atoms in total) from the logical hidden Markov model shown in Figure 7.3. We measured the averaged running time in seconds per iteration for both the naïve algorithm and SAGEM's gradient approach to evaluate neighbours when applied to the logical hidden Markov model shown in Figure 7.1. The times were measured using YAP's built-in `statistics/2`. The results are summarized in Figure 7.2 showing the ratio of running times of naïve over SAGEM's gradient approach. In some cases the speed-up ratio was more than 400. EM's lowest running time was 0.075 seconds (for 10 sequences of length 10). In contrast, SAGEM was constantly below 0.017 seconds. This suggests that **H1** holds.

We sampled 2000 sequences of length 15 (30000 ground atoms) from the logical hidden Markov model in Figure 7.3. There were 4 filenames and 2 users. The initial hypothesis was the logical hidden Markov model in Figure 7.1 with randomly initialized parameters. We run SAGEM on the sampled data. [32] Averaged over 5 runs, estimating the parameters for the initial hypothesis achieved a score of $-47203$. In contrast, the score of SAGEM's selected model was $-26974$, which was even slightly above the score of the original logical hidden Markov model ($-30521$). This suggests that **H3** holds. Moreover, in all runs, SAGEM included the following rules:

$$\texttt{latex(A,B)} \xleftarrow{0.61:\texttt{emacs(A)}} \texttt{emacs(A,B)} \text{ and } \texttt{emacs(A,B)} \xleftarrow{0.48:\texttt{emacs(A)}} \texttt{latex(A,B)} \,.$$

These rules were not present in the initial model. This suggests that **H2** holds.

### 7.4.2 Experiments with Real-World Data

Finally, we applied SAGEM to the data set collected by Greenberg [1988]. The data consists of 168 users of four groups: computer scientists, non-programmers, novices and others. About 300000 commands have been logged in on average 110 sessions per

[32] The naïve algorithm was no longer used for comparison due to unreasonable running times.

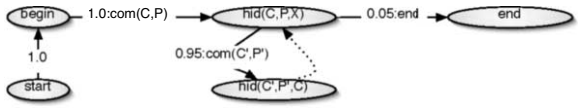



**Figure 7.4.** The initial hypothesis for the experiments with real-world data is a minimal structure, implying learning from scratch. $C$ stands for command and $P$ for parameters. The hidden state $\texttt{hid}(\texttt{C},\texttt{P},\texttt{L})$ contains the current command $\texttt{C}$, its parameters $\texttt{P}$, and the last command $\texttt{L}$.

user. We encoded the commands as sequences over $\texttt{com}(\texttt{C},\texttt{P})$ where $\texttt{C}$ is the command types in by the user and $\texttt{P}$ a list of parameters. For instance, $\texttt{cmd}(\texttt{ls},[]),\texttt{cmd}(\texttt{cd},['..'])$ denotes a user reading the directory, `ls`, and then changing to the parent directory, `cd` .. We present here results for two classes: novice-1(NV) consisting of 2512 ground atoms and non-programmers-4 (NP) consisting of 5183 ground atoms. We randomly selected 35 training sessions (about 1500 commands, i.e., ground atoms) for each class. On this data, we let SAGEM select a model for each class independently, starting from the initial hypothesis

$$1.00 \;:\; \texttt{begin} \leftarrow \texttt{start} \qquad\qquad 1.0 \;:\; \texttt{hid}(\texttt{C},\texttt{P},\texttt{X}) \xleftarrow{\texttt{com}(\texttt{C},\texttt{P})} \texttt{begin}$$

$$0.95 \;:\; \texttt{hid}(\texttt{C}',\texttt{P}',\texttt{C}) \xleftarrow{\texttt{com}(\texttt{C}',\texttt{P}')} \texttt{hid}(\texttt{C},\texttt{X},\texttt{Y}) \qquad 0.05 \;:\; \texttt{end} \xleftarrow{\texttt{end}} \texttt{hid}(\texttt{X},\texttt{Y},\texttt{Z}),$$

which is graphically represented in Figure 7.4. To evaluate, we computed the plug-in estimates, see Equation (6.3), of each model for the remaining sessions. Averaged over five runs, the precision ($0.94 \pm 0.06$ NV, $0.91 \pm 0.02$ NP) and recall values ($0.67 \pm 0.03$ NV, $0.89 \pm 0.05$ NP) were balanced and the overall predictive accuracy was $0.92 \pm 0.01$. Jacobs and Blockeel [2003] report that a kNN approach achieved a precision of 0.91 and J48 (WEKA's implementation of Quinlan's C4.5 decision tree learner) of 0.86 averaged over ten runs on 50 randomly sampled training examples. This suggests that **H4** holds.

The used kNN and J48 methods, however, do not yield generative models and lack comprehensibility. SAGEM's selected models encoded e.g. "*non-programmers are very likely to type in* `cd` .. *after performing* `ls` *in some directory*". This pattern was not present in the NV model. This suggests that **H2** holds.

# Future Work

The three fundamental problems of evaluating the likelihood of an observation sequence, estimating an optimal state sequence for observations, and learning the model parameters, all have quadratic time complexity in the number of the Herbrand base. Therefore, faster and approximate inference techniques have to be explored in the future. Logical hidden Markov models over bounded depth atoms (such as functor-free models) have a close connection to hierarchical and abstract HMMs. Therefore,

representing them as dynamic Bayesian networks [Siddiqi and Moore, 2005] and applying variational methods [Johns and Mahadevan, 2005] are promising directions for future research. In general, expressing logical hidden Markov models within more expressive probabilistic ILP languages such as PRISM programs would pave the way towards a rich toolbox of possible selection probability models including Bayesian networks and Markov chains. Furthermore, for large-state space models, it is usually reasonable to focus on a few high-probability transitions per state and keep track of their exact values while approximating the rest with a constant. This approach has been successfully applied to HMMs [Siddiqi and Moore, 2005] and is an attractive future approach to speed up inference within logical hidden Markov models. One might even use probability estimation trees (PET) to compactly and approximatively represent the forward and backboard probabilities. PETs might also be used to speed up structure learning. Instead of following a top-down hill-climbing approach using ILP-like refinement operators, one could induce a relational PET in each iteration of sagEM. The PET would represent the subsumption lattice among the abstract states. A similar technique has been used to learn RMMs [Anderson et al., 2002].

Finally, many interesting real-world applications are composed of multiple interacting processes, and thus merit a compositional representation of two or more random variables. This is typically the case for systems that have structure both in time and space. With a single state variable, Markov models and, in turn, logical Markov models are ill-suited for tackling these problems. Coupling multiple logical hidden Markov models or factorial variants of logical hidden Markov models with a distributed state space representation could capture these interactions. Another limit of the current framework of logical hidden Markov models are the restrictive from of selection distribution. More expressive selection distributions such as Bayesian networks or Markov chains, even encoded as neural networks, could be explored in the future. Furthermore, logical hidden Markov models are trained in an unsupervised manner. In many fields such as bioinformatics and computational linguistics, however, the task is to label each element of a given sequence. As an example, consider the task of labeling the words in a sentence with their corresponding part-of-speech (POS) tags. Each word is labeled with a tag indicating its appropriate part of speech. Thus, we are faced with a sequential supervised learning problem. Promising candidates are relational and logical variants of input-output HMMs and of conditional random fields. Recently, Gutmann [2005] applied conditional random fields to logical sequences. The idea is to represent cliques using logical regression trees. On a similar data set as we used in our protein fold classification task, Gutmann's method achieved a cross-validated accuracy of about 88%.

# Conclusions

Logical hidden Markov models, a new formalism for representing probability distributions over sequences of logical atoms, have been introduced and solutions to the three

central inference problems (evaluation, most likely state sequence and parameter estimation) have been provided. Experiments have demonstrated that unification can improve generalization accuracy, that the number of parameters of a logical hidden Markov model can be an order of magnitude smaller than the number of parameters of the corresponding HMM, that the solutions presented perform well in practice and also that logical hidden Markov models possess several advantages over traditional HMMs for applications involving structured sequences. Furthermore, a model selection method for logical hidden Markov models called sagEM has been introduced. sagEM fits the probabilistic learning from traces setting and combines *generalized EM*, which optimizes parameters, with structure search for model selection using ILP refinement operators. Experiments show sagEM's effectiveness.

## Related Work

Logical hidden Markov models combine two different research directions. On the one hand, they are a framework for statistical relational learning respectively probabilistic ILP over time. On the other hand, they are related to several extensions of HMMs and probabilistic grammars.

Within statistical relational learning, most attention has been devoted to developing highly expressive formalisms, such as e.g. PCUP [Eisele, 1994], PCLP [Riezler, 1998], SLPs [Muggleton, 1996], PLPs [Ngo and Haddawy, 1997], RBNs [Jäger, 1997], PRMs [Friedman et al., 1999], PRISM [Sato and Kameya, 2001], BLPs [see Part I], DPRMs [Sanghai et al., 2003], and MLNs [Domingos and Richardson, 2004, Sanghai et al., 2005], see also Section 5 for more details. Moreover, only Sanghai et al. [2003, 2005] considered dynamic processes. Logical hidden Markov models can be seen as an attempt towards *downgrading* such highly expressive frameworks. Indeed, applying the main idea underlying logical hidden Markov models to non-regular probabilistic grammars, i.e., replacing flat symbols with atoms, yields — in principle — stochastic logic programs [Muggleton, 1996]. As a consequence, logical hidden Markov models represent an interesting position on the expressiveness scale. Whereas they retain most of the essential logical features of the more expressive formalisms, they seem easier to understand, adapt and learn. This is akin to many contemporary considerations in ILP.

In the second type of approaches, the underlying idea is to *upgrade* HMMs and probabilistic grammars to represent more structured state spaces. Hierarchical HMMs [Fine et al., 1998], abstract HMMs [Bui et al., 2002], factorial HMMs [Ghahramani and Jordan, 1997], and HMMs based on tree automata [Frasconi et al., 2002] decompose the state variables into smaller units. In hierarchical HMMs, states emit sinlge symbols and abstract states emit strings of symbols. The strings emitted by abstract states are themselves governed by sub-HHMMs, which are called recursively. When the sub-HHMM is finished, i.e., the string is produced by a sequence of states, control is returned to wherever it was called from. The hierarchy has a bounded depth

so that hierarchical HMMs are less expressive than probabilistic context free grammars (PCFGs). Abstract HMMs apply the idea of hierarchical HMMs to explain the interaction between behaviors, or policies, at different levels of abstractions. Murphy and Paskin [2002] have shown how to interpret abstract HMMs as hierarchical HMMs. In factorial HMMs, states are be factored into $k$ state variables, which depend on one another only through the observation; and in tree based HMMs, the represented probability distributions are defined over tree structures. The key difference with logical hidden Markov models is that these approaches do not employ the logical concepts of variables and unification. Both are essential because variables allows one to group states together — thus, for instance in contrast to hierarchical HMMs, abstract states are not governed by logical HMMs and functors allow hierarchies of unlimited depth — and unification allows one to share knowledge between abstract states via abstract transitions. As our experimental evidence shows, sharing information among abstract states by means of unification can lead to more accurate model estimation. The same holds for *relational Markov models* (RMMs) [Anderson et al., 2002] to which logical hidden Markov models are closely related. In RMMs, states can be of different types, with each type described by a different set of variables. The domain of each variable can be hierarchically structured. The main differences between logical hidden Markov models and RMMs are that RMMs neither support variable binding nor unification nor hidden states.

The equivalent of HMMs for context-free languages are PCFGs. Like HMMs, they do not consider sequences of logical atoms and do not employ unification. Nevertheless, there is a formal resemblance between the Baum-Welch algorithms for logical hidden Markov models and for PCFGs. In case that a logical hidden Markov model encodes a PCFG both algorithms are identical from a theoretical point of view. They re-estimate the parameters as the ratio of the expected number of times a transition (resp. production) is used and the expected number of times a transition (resp. production) might have been used. The proof of Theorem 6.1 assumes that the PCFG is given in Greibach normal form[33] (GNF) and uses a pushdown automaton to parse sentences. For grammars in GNF, pushdown automata are common for parsing. In contrast, the actual computations of the Baum-Welch algorithm for PCFGs, the so called Inside-Outside algorithm [Baker, 1979, Lari and Young, 1990], is usually formulated for grammars in *Chomsky normal form*[34]. The Inside-Outside algorithm can make use of the efficient CYK algorithm [Hopcroft and Ullman, 1979] for parsing strings.

An alternative to learning PCFGs from strings only is to learn from more structured data such as *skeletons*, which are derivation trees with the nonterminal nodes removed [Levy and Joshi, 1978]. Skeletons are exactly the set of trees accepted by *skeletal tree automata* (STA). Informally, an STA, when given a tree as input, processes the tree bottom up, assigning a state to each node based on the states of that node's children. The STA accepts a tree iff it assigns a final state to the root

[33] A grammar is in GNF iff all productions are of the form $\mathtt{A} \leftarrow \mathtt{a}V$ where $\mathtt{A}$ is a variable, $\mathtt{a}$ is exactly one terminal and $V$ is a string of none or more variables.

[34] A grammar is in CNF iff every production is of the form $\mathtt{A} \leftarrow \mathtt{B}, \mathtt{C}$ or $\mathtt{A} \leftarrow \mathtt{a}$ where $\mathtt{A}, \mathtt{B}$ and $\mathtt{C}$ are variables, and $\mathtt{a}$ is a terminal.

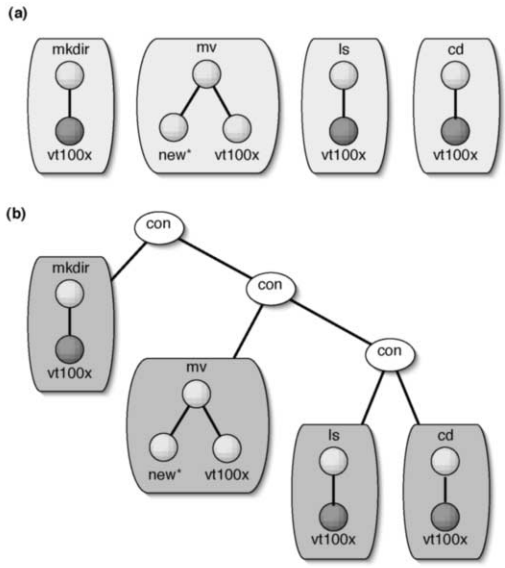



**Figure 7.1. (a)** Each atom in the logical sequence `mkdir(vt100x)`, `mv(new*, vt100x)`, `ls(vt100x)`, `cd(vt100x)` forms a tree. The shaded nodes denote shared labels among the trees. **(b)** The same sequence represented as a single tree. The predicate `con`/2 represents the concatenation operator.

of the tree. Due to this automata-based characterization of the skeletons of derivation trees, the learning problem of (P)CFGs can be reduced to the problem of an STA. In particular, STA techniques have been adapted to learning tree grammars and (P)CFGs [Sakakibara, 1992, Sakakibara et al., 1994] efficiently.

PCFGs have been extended in several ways. Most closely related to logical hidden Markov models are *unification-based grammars*, which have been extensively studied in computational linguistics. Examples are (stochastic) attribute-value grammars [Abney, 1997], probabilistic feature grammars [Goodman, 1997], head-driven phrase structure grammars [Pollard and Sag, 1994], and lexical-functional grammars [Bresnan, 2001]. For learning within such frameworks, methods for undirected graphical models are used; see [Johnson, 2003] for a description of some recent work. The key difference to logical hidden Markov models is that only nonterminals are replaced with structured, more complex entities. Thus, observation sequences of flat symbols instead of sequences of atoms are modelled. Goodman's *probabilistic feature grammars* are an exception. They treat terminals and nonterminals as vectors of features. No abstraction is made, i.e., the feature vectors are ground instances, and unification can not be employed. Therefore, the number of parameters that needs to be estimated becomes easily very large, data sparsity is a serious problem. Goodman applied smoothing to overcome the problem.

Logical hidden Markov models are generally related to (stochastic) tree au-

tomata, see e.g. [Carrasco et al., 2001]. Reconsider the UNIX command sequence `mkdir(vt100x), mv(new*, vt100x), ls(vt100x), cd(vt100x)`. Each atom forms a tree, see Figure 7.1 **(a)**, and, indeed, the whole sequence of atoms also forms a (degenerated) tree, see Figure 7.1 **(b)**. Tree automata process single trees vertically, e.g., bottom-up. A state in the automaton is assigned to every node in the tree. The state depends on the node label and on the states associated to the siblings of the node. They do not focus on sequential domains. In contrast, logical hidden Markov models are intended for learning in sequential domains. They process sequences of trees horizontally, i.e., from left to right. Furthermore, unification is used to share information between consecutive sequence elements. As Figure 7.1 **(b)** illustrates, tree automata can only employ this information when allowing higher-order transitions, i.e., states depend on their node labels and on the states associated to predecessors $1, 2, \ldots$ levels down the tree.

Finally, SAGEM is related to more advanced HMM model selection methods. *Model merging* [Stolcke and Omohundro, 1993] starts with the most specific model consistent with the training data and generalizes by successively merging states. For abstract transitions, however, the most general ones can be considered to be the most informative ones. Therefore, *successive state splitting* [Ostendorf and Singer, 1997] refines hidden states by splitting them into new states. In both cases, the authors do not employ Friedman's SEM.

This page intentionally left blank

Intermezzo

# Exploiting Probabilistic ILP in Discriminative Classifiers

Many real world applications can be regarded as classification problems: One tries to estimate the dependence $\mathbf{P}(Y|\mathbf{X})$ of a class variable $Y$ on some features $\mathbf{X}$, based on a finite set of observations $\mathbf{x}$ for which the value $y$ of the class variable is known. In Parts I and II, we developed generative probabilistic ILP approaches, which estimate the full joint distribution $\mathbf{P}(\mathbf{X}, Y)$ – hence *generative* – by learning the class prior distribution $\mathbf{P}(Y)$ and the class-conditional feature distribution $\mathbf{P}(\mathbf{X}|Y)$. The required posterior distribution is then obtained using Bayes' rule:

$$P(Y = y|\mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{X} = \mathbf{x}, Y = y)}{P(\mathbf{X} = \mathbf{x})} = \frac{P(\mathbf{X} = \mathbf{x}|Y = y) \cdot P(Y = y)}{\sum_{y'} P(\mathbf{X} = \mathbf{x}|Y = y) \cdot P(Y = y')} \ .$$

It is, however, well-known that the classification performance of generative models estimated from a finite set of observations is often lower than that of discriminative classifiers, which estimate $\mathbf{P}(Y|\mathbf{X})$ directly. Roughly speaking, the reason is that discriminative classifiers do not have to explain the values of the $\mathbf{X}$ features during learning.

One approach to improve the classifications performance based on generative models is to combine them with discriminative learners. *Fisher kernels* [Jaakkola and Haussler, 1999] were developed to combine generative models with a currently very popular class of learning algorithms, kernel methods. The key idea is to use the gradient of the log likelihood of the generative model with respect to its parameters as features because this captures the generative process rather than just the posterior probabilities. This Intermezzo showes that Fisher kernels naturally generalize to relational domains.

This page intentionally left blank

§ 8

# Relational Fisher Kernels *

*... in which, after briefly reviewing discriminative learning and kernel methods, Fisher kernels for interpretations and logical sequences are designed, and experiments show that these relational Fisher kernels can considerably improve the predictive performance of their probabilistic counterparts ...*

From a machine learning perspective, many real world applications can be regarded as classification problems: One tries to estimate the dependence of a target variable $Y$ on some observation $\mathbf{X}$, based on a finite set of observations $\mathbf{x}$ for which the value $y$ of the target variable is known. More formally, the classification problem can be defined as follows:

**Definition 8.1** (Classification Problem) **Given** a finite set of training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^m \subseteq \mathcal{X} \times \mathcal{Y}$ , where $\mathcal{X}$ is the feature space and $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$ is the set of possible classes, **find** a function $f : \mathcal{X} \to \mathcal{Y}$ with low approximation error on the training data as well as on unseen examples. ∘

**Example 8.2** Consider the KDD Cup 2001 localization of genes/proteins task [Cheng et al., 2002]. For every protein/gene, one wants to predict the localization of a protein/gene based on the features of the protein/gene and of proteins/genes interacting with the protein/gene. As another example from computational biology, recall the protein fold recognition *logical hidden Markov models* from Chapter 6, where one tries to understand how proteins fold up in 3D space. Usually, for every protein with unknown structure one aims at finding the most similar protein (fold) with known structure in a database. ∘

Recall that it is common to use the *plug-in estimate* for classification when using generative[35] models such as *logical hidden Markov models* (see e.g. Section 6.5) or *Bayesian logic programs*:

$$f(\mathbf{x}) = \arg\max_{y_i \in \mathcal{Y}} \frac{P(\mathbf{X} = \mathbf{x}, Y = y_i | \boldsymbol{\lambda}^*)}{P(\mathbf{X} = \mathbf{x} | \boldsymbol{\lambda}^*)} \tag{8.1}$$

$$= \arg\max_{y_i \in \mathcal{Y}} P(\mathbf{X} = \mathbf{x} | Y = y_i, \boldsymbol{\lambda}^*) \cdot P(Y = y_i | \boldsymbol{\lambda}^*) \ , \tag{8.2}$$

where $\boldsymbol{\lambda}^*$ are the maximum likelihood parameters of the given generative model. The maximum likelihood parameters are usually estimated using the EM algorithm. The predictive performance, however, of the plug-in estimate is often lower than that of discriminative classifiers.

* Builds on [Kersting and Gärtner, 2002, 2004, Dick and Kersting, 2006].

[35] They are called *generative* because they model the full joint probability distribution $\mathbf{P}(\mathbf{X}, Y)$by estimating $\mathbf{P}(\mathbf{X}|Y)$ and $\mathbf{P}(Y)$. Consequently, they can be used to sample, i.e., generate examples.

**Definition 8.3** (Discriminative Learner) A discriminative learner estimates the classification function $f : \mathcal{X} \to \mathcal{Y}$ of Definition 8.1 directly without representing the full joint probability distribution $\mathbf{P}(\mathbf{X}, Y)$. ◦

To improve the classification accuracy of generative models, different kernel functions have been proposed to make use of the good predictive performance of kernel methods such as support vector machine (SVM) [Schölkopf and Smola, 2002]. The most prominent of these kernel functions is the *Fisher kernel* [Jaakkola and Haussler, 1999, Jaakkola et al., 1999]. The key idea there is to use the gradient of the log likelihood of the generative model with respect to its parameters as features. The motivation to use this feature space is that the gradient of the log-likelihood with respect to the parameters of a generative model captures the generative process rather than just the posterior probabilities.

Fisher kernels have successfully been applied in many learning problems where the instances are described in terms of attribute-value vectors. So far, however, they have not been applied to relational data. As the term 'statistical relational learning' already says and as argued before, many real-world exhibit relational structure. Therefore, we suggest in this Intermezzo *relational Fisher kernels*.

**Definition 8.4** (Relational Fisher Kernel) Relational Fisher kernels are the family of kernel functions $k$ obtained by using the gradient $U_{\mathbf{x}} = \nabla_{\boldsymbol{\lambda}} \log P(\mathbf{X} = \mathbf{x} \mid \boldsymbol{\lambda}^*, M)$ of the log likelihood of a statistical relational learning model with respect to the model's parameters as features. ◦

We will experimentally show the benefits of relational Fisher kernels. More precisely, we will show that the predictive accuracy of Bayesian logic programs and logical hidden Markov models can considerably be improved using Fisher kernels and SVMs.

Before reviewing kernel methods in the next section, we would like to stress that, although we focus here on classification, methods for computing the gradient of the likelihood with respect to the parameters of probabilistic-logical models are of general interest as they can be used to devise fast gradient-based methods and accelerated EM algorithms for parameter estimation, see for instance [Salakhutdimov et al., 2003, Fischer and Kersting, 2003].

## 8.1 Kernel Methods and Probabilistic Models

Support vector machines [Schölkopf and Smola, 2002] are one kernel method that can be applied to binary supervised classification problems. Being on one hand theoretically well founded in statistical learning theory, they have on the other hand shown good empirical results in many applications.

The characteristic aspect of this class of learning algorithms is the formation of hypotheses by linear combination of positive-definite kernel functions 'centered' at individual training examples. It is known that such functions can be interpreted as the inner product in a Hilbert Space. The solution of the support vector machine is then the hyperplane in this Hilbert space that separates positive and negative labeled examples, and is at the same time maximally distant from the convex hulls of the

positive and the negative examples. Conversely, every inner product in a linear space is a positive-definite kernel function.

Fisher kernels are derived from a generative probability model of the domain. More precisely, every learning example is mapped to the gradient of the log likelihood of the generative model with respect to its parameters. The kernel is then the inner product of the examples' images under this map. The motivation to use this feature space is that the gradient of the log-likelihood with respect to the parameters of a generative model captures the generative process of a sequence better than just the posterior probabilities.

Given a parametric probability model $M$ with parameters $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)^\top$, maximum likelihood parameters $\boldsymbol{\lambda}^*$, and output probability $P(\mathbf{X} = \mathbf{x} \mid \boldsymbol{\lambda}, M)$, the Fisher score mapping $U_\mathbf{x}$ is defined as

$$\begin{aligned} U_\mathbf{x} &= \nabla_{\boldsymbol{\lambda}} \log P(\mathbf{X} = \mathbf{x} \mid \boldsymbol{\lambda}^*, M) \\ &= \left( \frac{\partial \log P(\mathbf{X} = \mathbf{x} \mid \boldsymbol{\lambda}^*, M)}{\partial \lambda_1}, \ldots, \frac{\partial \log P(\mathbf{X} = \mathbf{x} \mid \boldsymbol{\lambda}^*, M)}{\partial \lambda_n} \right)^\top . \end{aligned}$$

The Fisher information matrix is the expectation of the outer product of the Fisher scores over $P(\mathbf{X} = \mathbf{x} \mid \boldsymbol{\lambda}, M)$, more precisely,

$$J_{\boldsymbol{\lambda}} = E_\mathbf{x} \left[ \nabla_{\boldsymbol{\lambda}} \log P(\mathbf{x} \mid \boldsymbol{\lambda}, M) \right] \left[ \nabla_{\boldsymbol{\lambda}} \log P(\mathbf{x} \mid \boldsymbol{\lambda}, M) \right]^\top .$$

Given these definitions, the Fisher kernel is defined as

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= U_\mathbf{x}^\top J_{\boldsymbol{\lambda}^*}^{-1} U_{\mathbf{x}'} \\ &= \left[ \nabla_{\boldsymbol{\lambda}} \log P(\mathbf{X} = \mathbf{x} \mid \boldsymbol{\lambda}^*, M) \right]^\top J_{\boldsymbol{\lambda}^*}^{-1} \left[ \nabla_{\boldsymbol{\lambda}} \log P(\mathbf{X} = \mathbf{x}' \mid \boldsymbol{\lambda}^*, M) \right] . \end{aligned} \quad (8.3)$$

In practice often the role of the Fisher information matrix $J_{\boldsymbol{\lambda}}$ is ignored, yielding the kernel $k(\mathbf{x}, \mathbf{x}') = U_\mathbf{x}^\top U_{\mathbf{x}'}$. In the remainder of the Intermezzo we will follow this habit mainly to reduce the computational complexity. Learning algorithms using the Fisher kernel can be shown to perform well if the class variable is contained as a latent variable in the probability model. Jaakkola and Haussler [1999] have shown that under this condition (such as instances independently sampled from identical distributions) kernel machines using the Fisher kernel are asymptotically at least as good as choosing the maximum a posteriori class for each instance based on the model. Our experiments will indicate that the same holds in the case of relationally structured instances.

## 8.2 Fisher Kernels for Interpretations and Logical Sequences

According to Equation (8.3), it is sufficient to compute the gradient of the log likelihood of a data case with respect to the parameters $\boldsymbol{\lambda}$ of a probabilistic ILP model in order to devise a Fisher Kernel. In the following, we will derive Fisher kernels for interpretations respectively for logical sequences based on Bayesian logic programs respectively logical hidden Markov models. Nevertheless, we would like to stress the schematic nature of relational Fisher kernels. Any other probabilistic ILP framework appropriate for the type of data at hand can be used.

### 8.2.1 Fisher Kernels for Interpretations

Fisher kernels for interpretations can be devised using Bayesian logic programs. In Section 4.3.2, we have shown how to compute their gradient. As a reminder, consider a Bayesian program $M$ consisting of Bayesian clauses $c_i$ with $\mathrm{cpd}(c_i)_{jk} = P(u_j \mid \mathbf{u}_k)$ where $u_j \in \mathrm{D}(\mathrm{head}(c))$ and $\mathbf{u}_j \in \mathrm{D}(\mathrm{body}(c))$, and a single data case $D$. The $\mathrm{cpd}(c_i)_{jk}$'s constitute the parameter vector $\boldsymbol{\lambda}$ of $M$. Assuming decomposable combining rules, see Section 4.3.2, the partial derivative of the log-likelihood with respect to a parameter $\lambda$ of $\boldsymbol{\lambda}$ is

$$\frac{\partial \log P(D|\boldsymbol{\lambda}, M)}{\partial \lambda} = \sum_{\substack{\text{subst. } \theta \text{ with} \\ \text{support}(c_i\theta)}} \frac{P_N(\mathrm{head}(c_i\theta) = u_j, \mathrm{body}(c_i\theta) = \mathbf{u}_k \mid D)}{\mathrm{cpd}(c_i\theta)_{jk}} \tag{8.4}$$

where $P_N$ denotes the probability distribution of the support network induced by $M$ for data case $D$. Note that, in contrast to parameter estimation, we do not reparameterize the Bayesian logic program.

In many cases, it is difficult — if not impossible — to devise a generative Bayesian logic program specifying a probability distribution, which sums up to one over all possible instances, say proteins. For example in our experiments, examples are partly specified within the logical background knowledge. Consequently, their probabilities do not sum up to one and Equation (8.4) is sensitive to the number of contributing ground clauses. Normalizing (8.4) with respect to the number of contributing ground clauses, i.e., to compute

$$\frac{1}{|\{\theta|\,\mathrm{support}(c_i\theta)\}|} \sum_{\substack{\text{subst. } \theta \text{ with} \\ \text{support}(c_i\theta)}} \frac{P_N(\mathrm{head}(c_i\theta) = u_j, \mathrm{body}(c_i\theta) = \mathbf{u}_k \mid D)}{\mathrm{cpd}(c_i\theta)_{jk}} \,. \tag{8.5}$$

worked well in our experiments.

Equation (8.5) is all we need to devise Fisher kernels for interpretations. Therefore, we will now turn over to Fisher kernels for logical sequences. To highlight that the method can also be used for parameter estimation of logical hidden Markov models, we will derive the gradient of the likelihood of multiple observation sequences. The Fisher kernel is the special case of a single observation sequence only.

### 8.2.2 Fisher Kernels for Logical Sequences

Let $M$ be a logical hidden Markov models with parameters $\boldsymbol{\lambda}$ and $\mathsf{O} = \{\mathtt{O}_1, \ldots, \mathtt{O}_m\}$ be a set of ground observation sequences. A single ground observation sequence $\mathtt{O}_i$ consists of a sequence $\mathtt{o}_{i1}, \mathtt{o}_{i2}, \ldots, \mathtt{o}_{iT_i}$ of ground atoms. For the sake of simplicity, we assume that $T_1 = T_2 = \ldots = T_m$ and that the $\mathtt{O}_i$ are independently sampled from identical distributions (iid). Due to the iid assumption, it holds that

$$\frac{\partial \log P(\mathsf{O} \mid \boldsymbol{\lambda}, M)}{\partial \boldsymbol{\lambda}} = \sum_{k=1}^{m} \frac{\partial \log P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M)}{\partial \boldsymbol{\lambda}} = \sum_{k=1}^{m} \frac{1}{P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M)} \cdot \underbrace{\frac{\partial P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M)}{\partial \boldsymbol{\lambda}}}_{(*)} .$$

The key step to derive the gradient formula is to rewrite the likelihood $P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M)$ of a single observation sequence $\mathtt{O}_k$ in the following way:

$$\begin{aligned} P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M) &= \sum_{\mathtt{s} \in \mathbf{S}_t} P(\mathtt{o_{k1}}, \ldots, \mathtt{o_{kt}}, s_t = \mathtt{s} \mid \boldsymbol{\lambda}, M) \cdot P(\mathtt{o_{kt+1}}, \ldots, \mathtt{o_{kT_k}} \mid s_t = \mathtt{s}, \boldsymbol{\lambda}, M) \\ &= \sum_{\mathtt{s} \in \mathbf{S}_t} \alpha_t(\mathtt{s}) \cdot \beta_t(\mathtt{s}) \end{aligned} \tag{8.6}$$

where $\alpha_t(\mathtt{s})$ is the forward probability of state $\mathtt{s}$ and $\beta_t(\mathtt{s})$ is the backward probability of state $\mathtt{s}$ for $\mathtt{O}_k$. The term $S_t$ denotes the set of hidden states the system can be in at time $t$.

The parameter vector $\boldsymbol{\lambda}$ defines the set of parameters for all abstract transitions and for all selection distributions in the logical hidden Markov model. We will show now how to compute the partial derivatives $(*)$ for transition probabilities and for selection probabilities $\mu$ separately.

**Abstract Transitions** Let $\lambda_{ij}$ be an abstract transition probability, i.e., a probability value associated to the $j$th abstract transition

$$\mathtt{T} \equiv \lambda_{ij} : \mathtt{H} \overset{\mathtt{O}}{\leftarrow} \mathtt{B}$$

of the $i$th abstract body $\mathtt{B}$ in $\mathbf{B}$. Due to the chain rule it holds

$$\frac{\partial P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M)}{\partial \lambda_{ij}} = \sum_{t=0}^{T+1} \frac{\partial P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M)}{\partial \alpha_t(\mathtt{s_H})} \times \frac{\partial \alpha_t(\mathtt{s_H})}{\partial \lambda_{ij}}. \tag{8.7}$$

Due to Equation (8.6), the first term in the sum of Equation (8.7) is simply

$$\frac{\partial P(\mathtt{O}_k \mid \boldsymbol{\lambda}, M)}{\partial \alpha_t(\mathtt{s_H})} = \beta_t(\mathtt{s_H}) \tag{8.8}$$

because $\alpha_t(\mathtt{s_H})$ appears only in linear form. The partial derivative of $\alpha_t(\mathtt{s_H})$ w.r.t. $\lambda_{ij}$ in Equation (8.7) can be deduced from the *forward* procedure in Section 6.1:

$$\frac{\partial \alpha_t(\mathtt{s_H})}{\partial \lambda_{ij}} = \sum_{\mathtt{s_B} \in S_{t-1}} \xi(\mathtt{T}, \mathtt{s_B}, \mathtt{s_H}, \mathtt{o}_{kt-1}) \cdot \alpha_{t-1}(\mathtt{s_B}) \cdot \mu(\mathtt{s_H} \mid \mathtt{H}\sigma_{\mathtt{s_B}}) \cdot \mu(\mathtt{o}_{kt-1} \mid \mathtt{O}\sigma_{\mathtt{s_B}}\sigma_{\mathtt{s_H}}) \,,$$

where $\mathtt{o}_{kt-1}$ is the $t-1$-th observation symbol of the $k$-th observation sequence $\mathtt{O}_k$, $\xi(\mathtt{T}, \mathtt{s_B}, \mathtt{s_H}, \mathtt{o}_{kt-1})$ indicates that 1) $\mathtt{B}$ is maximally specific for $\mathtt{s_B}$, 2) $\mathtt{s_H}$ unifies with $\mathtt{H}$, and 3) $\mathtt{o}_{kt-1}$ unifies with $\mathtt{O}$, and $\sigma_{codt}$ are the corresponding MGUs.

**Selection Distribution** Now, let $\lambda_{ij}$ be a selection probability value. Let $\mathtt{r/n}$ be a predicate with domains $D_1, \ldots, D_n$, where $D_i = \{d_{i1}, \ldots, d_{im_i}\}$. Furthermore, assume that the the selection distribution for $\mathtt{r}$ is specified by $\lambda_{ij} = P(D_i = d_{ij})$. Equations (8.7) and (8.8) remain the same. The term $\partial \alpha_t(\mathtt{s})/\partial \lambda_{ij}$ is zero whenever

$d_{ij}$ was not *selected* to "ground" $\mathtt{H}\sigma_{\mathtt{b}}$ or $\mathtt{O}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}}$. Because, the selection distribution follows a naïve Bayes scheme and $\frac{\partial x^m}{\partial x} = m \cdot x^{m-1} = m \cdot \frac{x^m}{x}$, this yields:

$$\frac{\partial \alpha_t(\mathtt{s_H})}{\partial \lambda_{ij}} = \sum\nolimits_{\mathtt{s_B} \in S_{t-1}} \sum\nolimits_{\mathtt{T} \equiv p:\mathtt{H} \xleftarrow{0} \mathtt{B} \in \Delta} \xi(\mathtt{T}, \mathtt{s_B}, \mathtt{s_H}, \mathtt{o}_{kt-1}) \cdot c_{ij}(\mathtt{s_B}, \mathtt{s_H}, \mathtt{o}_{kt-1}) \cdot \\ \cdot \frac{\alpha_{t-1}(\mathtt{s_B}) \cdot p \cdot \mu(\mathtt{s_H} \mid \mathtt{H}\sigma_{\mathtt{s_B}}) \cdot \mu(\mathtt{o}_{kt-1} \mid \mathtt{O}\sigma_{\mathtt{s_B}}\sigma_{\mathtt{s_H}})}{\boldsymbol{\lambda}_{ij}} , \tag{8.9}$$

where $c_{ij}(\mathtt{T}, \mathtt{s_B}, \mathtt{s}, \mathtt{o}_{kt-1})$ denotes the number of times, the domain element $d_{ij}$ has been selected in order to ground $\mathtt{s_B}$, $\mathtt{s}$, and $\mathtt{o}_{kt-1}$ when following abstract transition $\mathtt{T}$.

### 8.2.3 Constraint Satisfaction

For parameter estimation of logical hidden Markov models, we have to take into account that the parameter vector consists of probability values. The same approaches as for Bayesian logic programs can be followed, see Section 4.3.2.

By now, we have everything together that is needed to compute the gradient of the log likelihood of Bayesian logic programs and of logical hidden Markov models with respect to their parameters, i.e., to compute relational Fisher kernels for interpretations and for logical sequences. In the next section, we will experimentally evaluate them.

## 8.3 Experimental Evaluation

Having described how to compute the gradient of the log likelihood of Bayesian logic programs and logical hidden Markov models with respect to their parameters, we are now ready to experimentally evaluate relational Fisher kernels. Our intention here is not to achieve high accuracies — although in some experiments this is the case — but rather to investigate the extent to which relational Fisher kernels are competitive with plug-in estimates:

**Q** Do relational Fisher kernels considerably improve the predictive accuracies of their probabilistic baselines with plug-in estimates?

We will split **Q** into two parts:

**Qa** Do SVMs with Fisher kernels based on Bayesian logic programs considerably improve the predictive accuracies of Bayesian logic programs with plug-in estimates?

**Qa** Do SVMs with Fisher kernels based on logical hidden Markov models considerably improve the predictive accuracies of logical hidden Markov models with plug-in estimates?

To investigate **Qa** and **Qb**, we compare results achieved by Bayesian logic programs respectively logical hidden Markov models alone with results achieved by Bayesian logic programs respectively logical hidden Markov models combined with Fisher kernels and SVMs.

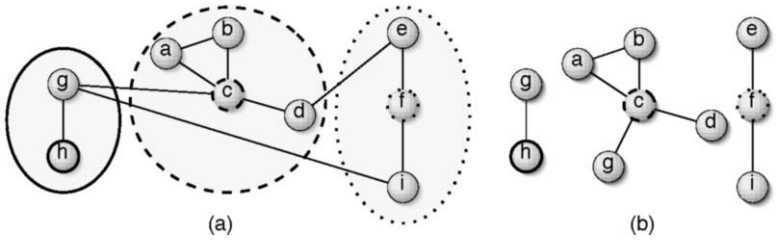



**Figure 8.1. (a)** A collective data set, i.e., a graph of connected individuals each described by a set of local features. **(b)** Data set broken into subgraphs centered around individuals. Each subgraph consists of an individual and all its direct neighbours. Individuals can appear in multiple fragments such as $g$.

### 8.3.1 Fisher Kernels for Interpretations

The experiments for **Qa** took place in two different domains: protein localization and web page classification. Both data sets are *collective* [Chakrabarti et al., 1998] or *networked* [Macskassy and Provost, 2004] data sets, i.e., relational data where individual examples are interconnected, such as web pages (connected through hyperlinks) or gene (connected through interactions). This contrasts with traditional relational domains such as molecules where each individual example is a graph of connected parts. Traditionally, machine learning methods treat examples as independent, i.e., the classification task is treated as a local optimization problem. In contrast, within collective classification tasks, the class membership of one individual may have an influence on the class membership of a related individual. Thus, learners developed for this task should treat the classification problem as a global optimization problem

There is a wide range of possible models that one can apply to the two tasks. We selected a set of models that we felt represented the main idea underlying a variety of collective learners [Chakrabarti et al., 1998, Neville and Jensen, 2000, Getoor et al., 2002, Lu and Getoor, 2003, Macskassy and Provost, 2004] who globally combine local, propositional Naïve Bayes classifiers. Relational Fisher kernels based on Bayesian logic programs, however, are not designed for collective classification [36]. They assume each individual example as a graph of connected parts. Therefore, we apply the following trick. While learning in a collective way, we consider only individuals together with their direct neighbours at classification time, cf. Figure 8.1. For any individual without any neighbours, we used a copy of the individual as neighbour. This is akin to *iterative* classifiers [Neville and Jensen, 2000, Macskassy and Provost, 2004], which also treat each individual together with all its direct neighbours as a single data case.

We investigated collective Naïve Bayes models and relational Fisher kernels derived from them as described above together with SVMs. The SVM algorithm used in our experiments was Weka's [Witten and Frank, 2005] implementation of Platt's [1998]

[36] Taking the whole graph at classification time would essentially yield the same feature vector for each individual because the data does not change.

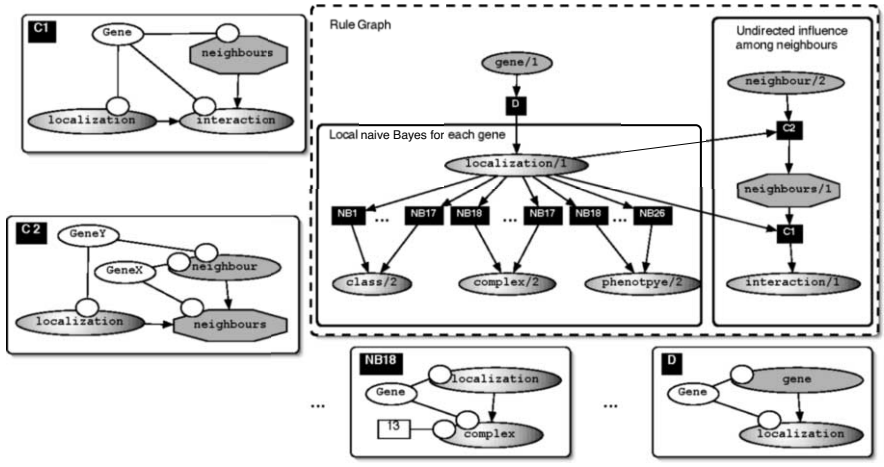



**Figure 8.2.** *Localization* Bayesian logic program. The Bayesian clauses $NB1, \ldots, NB17$ encode a Naïve Bayes over local features of genes `Gene`. Clause $D$ encodes the prior distribution over `localization` for each `gene`. Clause *C2* aggregates the localizations of all neighbouring genes of `Gene` in `neighbours`(`Gene`). The *mode* of the localizations is used as aggregate function. Finally, clause *C1* implements a mutual influence among `localization`(`Gene`) and the aggregated localization of interacting genes, `neighbours`(`Gene`).

sequential minimal optimization (SOM) algorithm using polynomial kernels. To reduce the number of features of the local Naïve Bayes models, we performed WEKA's greedy subset evaluation with default parameters on the training set. That is, we start with an empty feature set for the Naïve Bayes and add one feature on each iteration. If we have added all features or there is no improvement in score of the Naïve Bayes from adding any further features, the search stops and returns the current set of features. To score feature subsets, we used 10-fold cross-validated classification accuracy of the Naïve Bayes on the training set. Finally, both classification tasks are multiclass problems. In order to tackle multiclass problems with SVMs, we followed a round robin approach [Fürnkranz, 2002]. That is, each pair of classes is treated as a separate classification problem. The overall classification of an example instance is the majority vote among all pairwise classification problems.

**Protein Localization** The KDD Cup 2001 [Cheng et al., 2002] focused on data from life science. One data set, which we used in our experiments, is from genomics. The data consists of 1243 genes of one particular but unknown type of organism. Each gene encodes a protein, which occupies a particular position in some part of a cell. For each gene, information on the class, the phenotype, i.e., its characteristics, the complex it belongs to etc. are given. Furthermore, the graph of interactions among the genes is provided. The task was to predict the localization `locallization`(`Gene`) of a gene/protein `gene`(`Gene`). It is assumed that each gene/protein has only one

localization. In general, it can have more than one. This yields a multiclass problem with 16 different classes. 381 of the 1243 genes are withheld as test set.

Figure 8.2 shows the Bayesian logic program used in the experiments. We listed the genes as ground atoms over `gene`/1 in the logical background knowledge. They were used to encode the prior localization, cf. Bayesian clause $D$. The feature selection yielded 26 features for the local Naïve Bayes describing the genes, which we encoded as Bayesian clauses *NB1*, ..., *NB26*. So far, the Bayesian logic program encodes the simple, non-collective Naïve Bayes model we used in the experiments. To model the collective nature of the data set, we enriched the Naïve Bayes model as follows. We encoded each interaction as a logical ground atom over `direct_neighbour`/2, i.e., we omitted the originally given quantification of the interactions. Because interactions are bidirectional, i.e., undirected, we additionally defined the symmetric closure

```
neighbour(GeneA, GeneB) :- direct_neighbour(GeneA, GeneB);
                           direct_neighbour(GeneB, GeneA).
```

as logical background. The localizations of the direct neighbours of a `Gene` are aggregated in clause *C2* into a single value `neighbours(Gene)` using the *mode* of the interactions. To establish a mutual influence among the localizations of a gene and its neighbours, we introduced a boolean random variable `interaction(Gene)`, which has `neighbours(Gene)` and `localization(Gene)` as parents, cf. clause *C1*. Setting the evidence for `interaction(Gene)` always to be `true` guarantees that both parents are never d-separated, hence, they are probabilistic dependent.

On the test set, the relational Fisher kernel achieved an accuracy of 72.89%, whereas the collective Naïve Bayes only achieved 61.66%, and outperformed Hayashi et al.'s KDD Cup 2001 winning nearest-neighbour approach that achieved a test set accuracy of 72.18% [Cheng et al., 2002].

**Web Page Classification** This dataset is based on the WebKB Project [Craven et al., 2000]. It consists of sets of web pages from four computer science departments, with each page manually labeled into 7 categories: course, department, faculty, project, staff, student or others. We excluded pages in the 'other' category from consideration and put them into the background knowledge. This yielded a multiclass problem with 6 different classes, 877 web pages, and 1516 links among the web pages.

Figure 8.3 shows the Bayesian logic program used in the experiments. It essentially follows the idea underlying the Bayesian logic program for the localization task, cf. Figure 8.2. The feature selection yielded 67 local for the local Naïve Bayes model (clauses *NB1*, ..., *NB67*. Whereas gene interaction is undirected, links among web pages are directed. There are *incoming* and *outcoming* links on a web page. We modeled their influences on the class membership of a web pages separately. The atom `neighbours_from(Page)` (`neighbours_to(Page)`) aggregates the class memberships of all pages that have a link to `Page` (that `Page` links to) using *mode* as aggregate function. Again, we took care that `class(Page)` and the aggregated class memberships of linked pages mutually influence each other, i.e., we introduced `isLinked_from(Page)` and `isLinked_to(page)`, whose evidence is always `yes`.

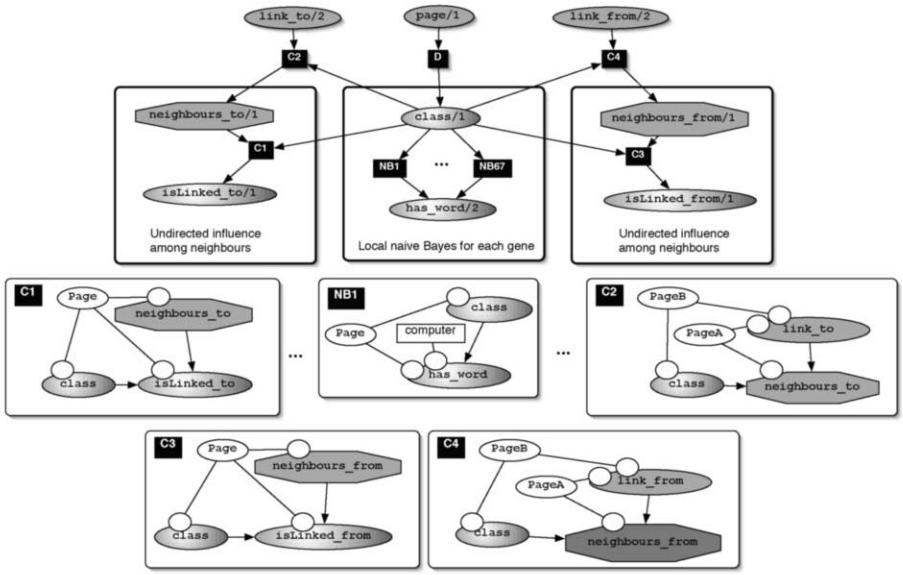
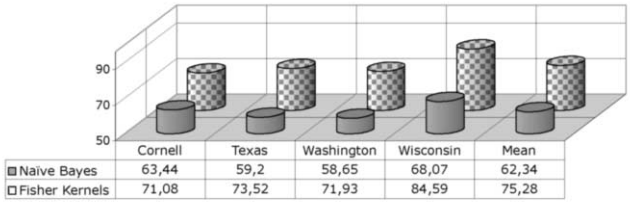



**Figure 8.3.** *WebKB* Bayesian logic program. The Bayesian clauses *NB1*, ..., *NB67* encode a Naïve Bayes over local features of genes web pages, `page(Page)`. Clause *D* encodes the prior distribution over `class` for each `Page`. Clause *C2* aggregates the class memberships of all web pages to which `Page` provides a link. Clause *C4* aggregates the class memberships of all web pages, which link to `Page`. In both cases, the *mode* of the class memberships is used as aggregate function. Finally, clauses *C1* and *C3* implement a mutual influence among `class(Page)` and the aggregated class memberships of linked pages.

We performed a leave-one-university-out cross-validation. The experimental results are summarized in Figure 8.4. The Fisher kernels achieved an accuracy of 75.28%, which is significantly higher (two-tailed t-test, $p = 0.05$) than the collective Naïve Bayes' accuracy of 62.34%. For comparison, the performance of the collective Naïve Bayes is in the range of Getoor et al.'s [2002] probabilistic relational model with link anchor words. The Fisher kernel outperforms the probabilistic relational model with the best predictive accuracy Getoor et al. report on. It takes structural uncertainty over the link relationship of web pages into account and achieved with 68% its highest accuracy on the Washington hold-out set. Thus, **Qa** is affirmatively answered.

### 8.3.2 Fisher Kernels for Logical Sequences

The experiments took place in the two bioinformatics domains already used to evaluate logical HMMs in Section 6.5: Protein fold recognition and mRNA signal structure detection. Both problems are multiclass problems with 5 different classes each. In order to tackle the multiclass problem with SVMs, we create for each class a binary classification problems, treating instances of this class as positive and all other instances as negative (one-against-all). As all binary classification problems consist of




| | Cornell | Texas | Washington | Wisconsin | Mean |
|---|---|---|---|---|---|
| Naïve Bayes | 63,44 | 59,2 | 58,65 | 68,07 | 62,34 |
| Fisher Kernels | 71,08 | 73,52 | 71,93 | 84,59 | 75,28 |

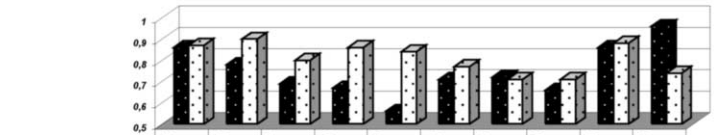
**Figure 8.4.** Leave-one-university-out accuracies on the WebKB data. Both collective classifiers used the same Bayesian logic program. The mean difference between collective Naíve Bayes and relational Fisher kernel in test accuracy was 12.94%.

the same instances and the SVMs on each classification problem were trained with the same parameters, the resulting models are comparable. That is, to create a multiclass classification we compare the numerical output of the binary support vector machines on each test instance, and assign the class corresponding to the maximal numerical output. The SVM algorithm used in our experiments was SVM-light [Joachims, 2002].

**mRNA Signal Structure Detection** This experiment is concerned with identifying subsequences in mRNA that are responsible for biological functions. In contrast to secondary structures of proteins that form chains (see next experiment), the secondary structures of mRNAs form tree structures. As trees can not easily be handled using HMMs, mRNA secondary structure data is more interesting than that of proteins.

The logical hidden Markov models we used are essentially the tree-structured logical HMMs used in the experiments reported in Section 6.5.3. The only difference is a lower number of copies for lengths in the sequence model, cf. Figure A.1 in Appendix A.2, namely 4. The leave-one-out cross-validated error of the plug-in estimates increased from 1% to 4.3%. Nevertheless, the error rate of 4.3% could be reduced to 2.2% by using Fisher kernels. As the dataset is rather small, we used leave-one-out error estimation and did not further optimize the SVM parameters. That is, we used a linear kernel and let SVM-light choose the default complexity constant. More precisely, the Fisher kernels managed to resolve two misclassifications, one of IRE and one of SECIS. The result improves the error rate of 4.6% reported in Horváth et al. [2001]. This is an affirmative answer to Question **Qb**.

**Protein Fold Recognition** This experiment is concerned with how proteins fold up in nature. This is an important problem, as the biological functions of proteins depend on the way they fold up. A common approach to protein fold recognition is to start from a protein with unknown structure and search for the most similar protein (fold) with known structure in the database. This approach has been followed in Section 6.5.2. Here, we will also compare Fisher kernels to the results earlier reported in Kersting et al. [2003b], where logical hidden Markov models with plugin estimates had been used. Notice that the number of parameters of the logical hidden Markov




| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
|---|---|---|---|---|---|---|---|---|---|---|
| | Fold1 | | Fold2 | | Fold 23 | | Fold37 | | Fold 55 | |
| LOHMMs | 0,86 | 0,78 | 0,69 | 0,67 | 0,56 | 0,71 | 0,72 | 0,66 | 0,86 | 0,96 |
| Fisher Kernels | 0,87 | 0,9 | 0,8 | 0,86 | 0,84 | 0,77 | 0,71 | 0,71 | 0,88 | 0,74 |

**Figure 8.5.** Precision and recall values of logical hidden Markov models and relational Fisher kernels for the protein fold classification task.

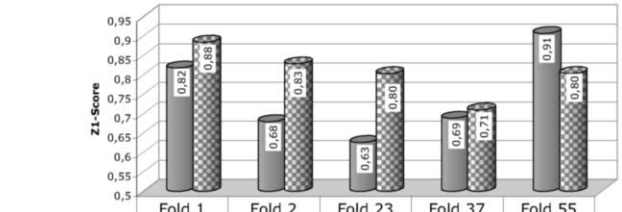
model structure used were by an order of magnitude smaller than the number of an equivalent HMM (120 vs. approx. 62000).

Here, we will compare to earlier results reported in [Kersting et al., 2003b], which are less elaborated and, hence, better show the benefit of using Fisher kernels. The data consists of logical sequences of the secondary structure of protein domains. The task is to predict one of five SCOP [Hubbard et al., 1997] folds for 2187 test sequences given a logical HMM trained on 200 training sequences per fold. As this dataset is lager we were able to perform a proper parameter selection. We first performed a leave-one-out error estimation in the training set to choose the parameter of a Gaussian kernel function. Of the tested parameters ($\gamma \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$), $\gamma = 10^{-3}$ clearly performed best over all binary problems. We then fixed this parameter and optimized the complexity constant. Of the tested parameters ($C \in \{10^{-1}, 10^{0}, 10^{-1}, 10^{2}\}$), $C = 100$ clearly performed best over all binary problems (testing larger values was not necessary, as we already achieved 0 unbounded support vectors).

Using Fisher kernels with the same logical hidden Markov as in [Kersting et al., 2003b] and the above described SVM parameters, we were able to reduce the error rate of the plugin estimate of 26% to an error rate of 17.4%. The logical hidden Markov models in Section 6.5.2 achieved a (cross-validated) error rate of 24% on a similar data set. The precision and recall values (precision/recall) were well balanced within each class as Figure 8.5 show. Finally, the $F_1$-scores were higher on average, see Figure 8.6. The $F_1$-score is the harmonic average of the precision and recall values: $F_1 = (2 \cdot \mathit{Precision} \cdot \mathit{Recall})/(\mathit{Precision} + \mathit{Recall})$ . This is an affirmative answer to question **Qb**.

## 8.4 Future Work and Conclusions

The research on kernel and discriminative, probabilistic ILP has just started and is in its infancy. In the last years, a number of powerful kernel-based machines such as support vector machines, kernel Fisher discriminant and kernel PCA have been proposed and proven to be relevant in practice. Choosing the appropriate




| | Fold 1 | Fold 2 | Fold 23 | Fold 37 | Fold 55 |
|---|---|---|---|---|---|
| LOHMMs | 0,8180488 | 0,6798529 | 0,6261417 | 0,6886957 | 0,9072527 |
| Fisher Kernels | 0,8847458 | 0,8289157 | 0,8034783 | 0,71 | 0,8039506 |

**Figure 8.6.** $F_1$-scores of logical hidden Markov models [Kersting et al., 2003b] and relational Fisher kernels for the protein fold classification task.

kernel is the major step for their application and should take as much as possible the available domain background knowledge into account. As Frasconi et al. [2005] point out, this is a major difficulty because real-valued functions are inappropriate as a knowledge representation language. Thus, they suffer — even more than Bayesian networks and hidden Markov models — from their non-declarative nature. Here, relational and first-order logical languages provide a natural alternative. The work on relational Fisher kernels provides only a first step to embed knowledge into kernel methods. We believe that there is a lot space for future research, For instance, stochastic versions of Passerini et al.'s [2005] trace kernels could be considered. The idea of trace kernels is to use *visitor* programs to generate proof trees of the examples, which exploit the available background knowledge, while set kernels [Shawe-Taylor and Cristianini, 2004] employ these proof trees for learning. Furthermore, other learning tasks could benefit from probabilistic relational kernels including regression, clustering, ranking, and novelty detection. Replacing the visitor program by a stochastic or Bayesian logic program is an obvious choice. One might also explore the connection between string and Fisher kernels [Saunders et al., 2003] for sequences of logical atoms. 'Logical string' kernels would be particularly interesting because subsumption for logical sequences is polynomial [Lee and De Raedt, 2004].

So far, Fisher kernels have been considered for flat data and for sequences of logical atoms only. In this Intermezzo, Fisher kernels for interpretations and for logical sequences have been introduced and experimentally investigated. They consider a discriminative setting for probabilistic ILP. The experimental results show that Fisher kernels can handle interpretations and logical sequences. More importantly, relational Fisher kernels can improve considerably the predictive performance of Bayesian logic programs and logical hidden Markov models.

## 8.5 Related Work

Discriminative learning has recently started to receive attention within statistical relational learning. To the best of our knowledge, Taskar et al. [2002, 2004a, 2005]

and Singla and Domingos [2005] are the only ones who aim at discriminative (probabilistic) models for structured data. In contrast to relational Fisher kernels, however, Taskar et al. [2002] and Singla and Domingos [2005] do not explore kernel functions but gradient-based optimization of the conditional likelihood $\mathbf{P}(y|\mathbf{x})$. Furthermore, the work by Taskar et al. [2002, 2004a, 2005] considers relational variants of Markov networks instead of Bayesian networks. Recently, Landwehr et al. [2005] and Davis et al. [2005] tightly integrated Naïve Bayes with ILP techniques focusing on discriminative objective functions such as conditional likelihood, accuracy and area under the precision-recall curve.

Indeed, there has been a lot interest in kernels for structured data in the kernel community. For an overview, we refer to [Gärtner, 2003, 2005]. In principle, there are two ways to apply support vector machines to structured data: Using *syntax-driven* and *model-driven* kernel functions.

An integral part of many *syntax-driven* kernels for structured data is the *decomposition* of an object into a set of its parts and the *intersection* of two sets of parts. The kernel on two objects is then defined as a measure of the intersection of the two corresponding sets of parts. In the case that the sets are finite or countable sets of vectors it is often beneficial to sum over all pairwise kernels on the elements. This idea of intersection and cross-product kernels is reflected in most work on kernels for structured data, from the early and influential technical reports [Haussler, 1999, Watkins, 1999], through work on string kernels, kernels for higher order terms [Gärtner et al., 2004], and tree kernels [Collins and Duffy, 2001], to more recent work on graph kernels [Gärtner et al., 2003, Kashima et al., 2003].

An alternative to syntax-driven kernel functions are *model-driven kernel* functions like Fisher kernel. Based on the idea of maximizing the posterior probability estimated by the optimal logistic regressor in the extracted feature space, Tsuda et al. [2002a] introduced the so-called TOP kernel function. The TOP kernel function is the scalar product between the posterior log-odds of the model and the gradient thereof. The posterior log-odds is defined as the difference in the logarithm of the probability of each class given the instance. A similar approach has been applied to speech recognition [Smith and Gales, 2002]. Marginalized kernels [Tsuda et al., 2002b] have later been introduces as a generalization of Fisher kernels. Here, a kernel over both the hidden and the observed data is assumed to be given. Then, the marginalized kernel for the visible data is obtained by taking the expectation with respect to hidden variables.

Part III

# Making Complex Decisions in Relational Domains

In this final part, we extend the capabilities of probabilistic ILP agents towards decision-theoretic reasoning. Such an agent begins within some knowledge of the world and of its own actions. It uses probabilistic ILP techniques to maintain a relational description of the world as new percepts arrive, and to deduce a course of actions that will achieve its goal. Due to uncertainties, it is not known what will happen in the future when actions are applied, i.e., actions are not guaranteed to achieve goals. Therefore, agents need ways to weighting up the utilities of goals and the likelihood of achieving these goals. Thus, relational decision-theoretic approaches can also be viewed as special-purpose probabilistic ILP approaches that reason efficiently with relational and logical axioms describing uncertain actions.

More precisely, Part III introduces *Markov decision programs* which combine Markov decision processes with logic programming in Chapter 9. Then, in Chapter 10, the *generalized relational policy iteration* scheme for solving Markov decision programs is developed. Following this scheme, two reinforcement learning approaches are devised: *relational TD*($\lambda$), a model-free approach, and *relational value iteration*, a model-based approach employing a relational Bellman backup operator, called ReBel. Convergence issues are addressed, and experimental results are presented as well.

This page intentionally left blank

§ 9

# Markov Decision Programs *

*... in which Markov decision processes are reviewed, the framework of Markov decision programs (MDPrgs) that combines Markov decision processes with logic programming is introduced, and the semantics of Markov decision programs are defined ...*

The ability of animals to learn appropriate actions in response to particular stimuli on the basis of associated rewards or punishment is a focus of behavioral psychology. In instrumental conditional, the actions of the animal determine what reinforcement is provided. Learning about stimuli or actions solely on the basis of the rewards and punishments associated with them is called reinforcement learning (RL) [Sutton and Barto, 1998]. It is the problem faced by an agent that must learn optimal behavior through trial-and-error interactions with a dynamic environment. More precisely, an agent acts in an environment and occasionally receives some reward based on the state the agent is in and the action(s) the agent took. The agent's learning task is to find a policy for action selection that maximizes its reward over the long term. This task requires not only choosing those actions that are associated with high rewards in the current state but also looking ahead by choosing actions that will lead the agent to more lucrative parts of the state space. Thus, in contrast to discriminative learning, see Definition 8.1, reinforcement learning is minimally supervised because agents are not told explicitly the actions to take in particular situations, but must work this out for themselves on the basis of the rewards they receive. Consider for instance an agent in a simulated blocks world.

**Example 9.1** The blocks world agent must learn to use its hand to remove blocks on some block `a` so that block `a` may be lifted. Only when block `a` is clear, i.e., no other block is on top of `a`, the agent receives a positive reward. Nevertheless, the agent has to decide on the utility of taking actions in each states of the world. ○

Many fascinating reinforcement learning techniques have been developed over the last few decades. Most of them use traditional statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world. In particular, most techniques assume propositional representations, which essentially amounts to enumerate all unique configurations of blocks. It might then be possible to learn, for example, that taking action `action234` in state `state42` is worth 6.2 and leads to state `state654321`. Propositional representations are simple (a single ground atom represents an entire state) and learning can be implemented using simple look-up tables. These look-up tables, however, can be intractably large, i.e., propositional representations easily explode. For instance for 4 blocks, there are 73 states, for 7 blocks 37,663 states, and for 10 blocks, there are already 58,941,091 states. Indeed, propositional states and actions can also be described using conjunctions of ground atoms.

* Builds on [Kersting and De Raedt, 2003, 2004, Kersting et al., 2004].

**Example 9.2** The conjunction $\texttt{cl(c)}, \texttt{on(c,b)}, \texttt{on(b,a)}, \texttt{block(a)}, \texttt{block(b)}, \texttt{block(c)}$ describes that there is no block on top of block `c`, block `c` is on block `b`, and block `b` is on `a`. ○

Though this representation affords some opportunities for generalization, we must still refer to objects by name (e.g., `block(a)` and `block(b)`). This prevents generalization over several states and actions such as $\texttt{on(X,b)}, \texttt{block(X)}, \texttt{block(b)}$.

Thus, reinforcement learning suffers from the same propositional nature and limitations, which we already encountered for Bayesian networks and logical hidden Markov models in Part I respectively Part II: no general situations and actions, i.e., no general regularities can be described. It is therefore not surprising that there has been an increased attention for dealing with relational representations and objects in reinforcement learning, see e.g. [Džeroski et al., 2001, Finney et al., 2002, Driessens, 2004]. Many of these works have taken a practical perspective and have developed systems and experiments that operate in relational worlds. At the heart of these systems there is usually a function approximator (often a logical regression tree inducer) that is able to learn a function assigning values to sets of states or to sets of state–action pairs. So far, however, a theory that explains why this approach works seems to be lacking.

Here, we will provide a first step towards a theory of relational reinforcement learning. Markov decision processes (MDPs) provide the theoretical foundations for traditional reinforcement learning, decision-theoretic planning, and other sequential decision-making tasks of interest to researchers and practitioners in artificial intelligence and operational research. Therefore, we will introduce a novel representation formalism, called *Markov decision programs* (MDPrgs), that combines Markov decision processes with *logic programming*. Markov decision programs are a flexible and expressive framework for defining MDPs that are able to handle structured objects as well as relations. They share several advantages with the other probabilistic ILP frameworks presented in this thesis. First, logical expressions (in the form of clauses, rules or transitions) may contain variables and as such make *abstraction* of many specific *grounded* states and transitions. Second, unification allows to elegantly share knowledge among sets of states. This allows one to compactly represent complex domains. Furthermore, because of this abstraction, the number of parameters (such as rewards and probabilities) is significantly reduced. This in turn allows one – in principle – to speed up and simplify the learning because one can learn at the *abstract* rather than at the *ground* level. For Markov decision processes, such a framework making abstraction through the use of logic programming concepts has been missing. Boutilier et al. [2001] reported on combining MDPs with Reiter's situation calculus. However, it is more complex, and Boutilier et al. did not consider model-free reinforcement learning techniques. The same holds for combinations of MDPs with the fluent calculus [Großmann et al., 2002, Karabaev and Skvortsova, 2005].

## 9.1 Markov Decision Processes

We will introduce Markov decision programs using examples from the *blocks world* [Slaney and Thiébaux, 2001], which is one of the standard planning domains.

**Example Domain III** (Blocks World) *The domain consists of a surface, called the floor, on which there are blocks. Blocks may be on the floor or on top of other blocks. They are said to pile up in stacks, each of which is on the floor. Valid relations are* `on(X,Y)`*, i.e., block* `X` *is on* `Y`*, and* `cl(Z)`*, i.e., block* `Z` *is* clear*. To model the* `floor`*, we follow a common approach. It is a set of blocks that cannot be on top of other blocks. At each time, the agent can move a clear (and movable) block* `X` *onto another clear block* `Y`*. The* `move`$(X,Y,Z)$ *action is probabilistic, i.e., it may not always succeed. For instance, with probability* $p_1$ *the action succeeds, i.e.* `X` *will be on top of* `Y`*. With probability* $1-p_1$*, however, the action fails. More precisely, with probability* $p_2$ *the block* `X` *remains at its current position, and with probability* $p_3$ *(with* $p_1+p_2+p_3=1$*) it falls on some clear block* `Z`*.*

*The blocks world has been a standard example in the literature on planning over the last decades and has extensively been used as a benchmark for domain-independent planning techniques and systems. Recently, the international planning competition (IPC-04) established a probabilistic planning track for the first time [Littman and Younes, 2004]. Half of the test problems were from blocks world.* ∘

A natural formalism to treat the utilities and uncertainties of the blocks world are Markov decision processes. A Markov decision process (MDP) is a tuple $\mathbf{M} = (S, A, \mathbf{T}, \lambda)$. Here, $S$ is a set of system states such as $z \equiv$ `cl(a),on(a,b),on(b,floor),block(a),block(b)` describing the blocks world consisting of two blocks `a` and `b` where `a` is on top of `b`. The agent has available a finite set of actions $A(z) \subseteq A$ for each state $z \in S$, which cause stochastic state transitions, for instance, `move(a,floor)` moving `a` on the `floor`. For each $z, z' \in S$ and $a \in A(z)$ there is a transition $T$ in $\mathbf{T}$, i.e., $z' \xleftarrow{p:r:a} z$. The transition denotes that with probability $P(z,a,z') := p$ action $a$ causes a transition to state $z'$ when executed in state $z$. For instance $z' \equiv$ `cl(a),cl(b),on(a,floor),on(b,floor),block(a),block(b)`. For each $z \in S$ and $a \in A(z)$ it holds $\sum_{z' \in S} P(z,a,z') = 1$. The agent gains an expected next reward $R(z,a,z') := r$ for each transition. Often, rewards are associated with states only, i.e., the agent gains a reward when entering a state, denoted as $R(z) := r$. In the blocks world we could have $R(z') = 10$. If the reward function $R$ is probabilistic (mean value depends on the current state and action only) the MDP is called *nondeterministic*, otherwise *deterministic*. We only consider MDPs with stationary transition probabilities and stationary, bounded rewards. A (stationary) deterministic policy $\pi : S \mapsto A$ is a set of expressions of the form $a \leftarrow z$ for each $z \in S$ where $a \in A(z)$, e.g. `move(a,floor)` ← `cl(a),on(a,b),on(b,floor),block(a),block(b)`. It denotes a particular course of actions to be adopted by an agent, with $\pi(z) := a$ being the action to be executed whenever the agent is in state $z$. We assume an infinite horizon and also that the agent accumulates the rewards associated with the states it enters.

Suppose now that the sequence of rewards after step $t$ is $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$ The agent's goal is to maximize the expected reward $E[R]$ for each step $t$. Typically, future rewards are discounted by $0 \leq \lambda < 1$ so that the expected return basically becomes $\sum_{k=0}^{\infty} \lambda^k \cdot r_{t+k+1}$. To achieve this, most techniques employ value functions. More precisely, given some MDP $M = \langle S, A, T, R \rangle$, a policy $\pi$ for $M$, and a *discount factor* $\gamma \in [0,1]$, the *state value function* $V^\pi : S \to \mathbb{R}$ represents the value of being

in a state following policy $\pi$ with respect to the expected reward. In other words, the value $V^\pi(z)$ of a state $z$ is the expected return starting from that state, which depends on the agent's policy $\pi$. A similar *state-action value function* $Q^\pi : S \times A \to \mathbb{R}$ can be defined. The value $Q^\pi(z, a)$ of taking action $a$ in state $z$ under policy $\pi$ is the expected return starting from that state, taking that action, and thereafter following $\pi$. Using the value functions, policies can be partially ordered. A policy $\pi'$ is better than or equal to another policy $\pi$, $\pi' \geq \pi$, if and only if $\forall s \in S \ : \ V^{\pi'}(s) \geq V^\pi(s)$. Thus, a policy $\pi^*$ is optimal, i.e., it maximizes the expected return for all states if $\pi^* \geq \pi$ for all $\pi'$. Optimal value functions are denoted $V^*$ and $Q^*$. Bellman's [1957] *optimality equation* states:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] \tag{9.1}$$

From this equation, basically all model-based and model-free methods for solving MDPs can be derived. For example, the well-known exact and model-based solution technique of *value iteration* (VI) is obtained from (9.1) by turning it into an update rule:

$$V_{t+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_t(s')] = \max_a Q_{t+1}(s, a). \tag{9.2}$$

Based on Equation (9.2), the VI algorithm can be stated as follows:

> starting with a value function $V_0$ over all states, we iteratively update the value of each state according to (9.2) to get the next value functions $V_t$ ($t = 1, 2, 3, \ldots$).

VI is guaranteed to converge in the limit towards $V^*$, i.e. the Bellman optimality equation (9.1) will hold for each state in the limit.

As an example for a model-free approach consider *temporal difference* (TD) learning methods. TD methods do not require a model of the environment's dynamics; instead they directly learn from experience. Like VI, TD methods bootstrap estimates, i.e., they update estimates based in part on already learned estimates. Starting with some initial value function $V_0$, at time $t + 1$ TD methods immediately make an update using the observed reward and the current estimate $V_t$. The simplest TD method, known as TD(0), uses the following update rule

$$V_{t+1}(z) = V_t(z) + \alpha[r_{t+1} + \gamma V_t(z') - V_t(z)] \tag{9.3}$$

Traditional MDPs as well as their solution methods such as VI (9.2) and TD(0) (9.3) are essentially propositional in that each state must be represented using a separated proposition. Therefore, they are severely limited in expressiveness and do not really capture the structure of the underlying class of problems. Recall that in the blocks world there are already $58,941,091$ states for ten blocks. A further consequence is that is hard to generalize policies across domains with similar properties, such as domains with a different number of blocks world states and/or tasks. Markov decision programs (MDPrgs) — which we will introduce in the next section — combine relational logic with MDPs. Using MDPrgs, it becomes possible to generalize such policies even for those cases where the states may posses a varying number of objects (blocks) and relations (`on`$/2$, `cl`$/1$) among them.

## 9.2 Representation Language

The *logical component* of a Markov decision process is essentially propositional because the state and action symbols employed are flat. Inspired by the idea of logical HMMs, cf. Part II, the key idea underlying *Markov decision programs* (MDPrgs) is to replace these flat symbols by abstract symbols. In contrast to logical HMMs, however, an abstract state is not a single atom only but a conjunction of atoms.

**Definition 9.3** An *abstract state* is a conjunction $Z$ of logical atoms, i.e., a logical query. ∘

An abstract state $Z$ represents a set $S(Z)$ of states. More formally, a state is a Herbrand interpretation, i.e., a set of ground facts and $Z$ represents the set $S(Z)$ of states, which are subsumed by $Z$.

**Example 9.4** Reconsider the blocks world state $z \equiv \mathtt{cl(a)}, \mathtt{cl(b)}, \mathtt{on(b,c)}$. An abstract state $Z$ is, e.g., $\mathtt{cl(X)}$. It represents all states that are subsumed by $Z$, i.e., all interpretations in which there exists an $\mathtt{X}$ that is clear. For instance, $z$ is subsumed by $Z$. ∘

We can now introduce the basic ingredients of Markov decision programs, namely, abstract actions, abstract rewards, and integrity constraints.

**Definition 9.5** An *abstract action*[37] is a finite set of abstract transitions $H_i \xleftarrow{p_i:A} B$ where $A$ is an atom representing the name and the arguments of the action and $B$ is an abstract state denoting the preconditions of $A$, $H_i$ is the $i$-th possible outcome of $A$, and $\sum_i p_i = 1$. ∘

We assume that $\mathrm{vars}(A) \subseteq (\mathrm{vars}(H_i) \cup \mathrm{vars}(B))$. The semantics of the action definition are as follows:

**Semantics 9.6** (Abstract Action) If state $b$ is subsumed by $B$ with substitution $\theta$, i.e., $b \preceq_\theta B$, then taking action $A\theta$ in $b$ will result in $[b \setminus B\theta] \cup H_i\theta$ with probability $p_i/|\theta|$ where $|\theta|$ is the number of possible substitutions $\theta$ such that $b \preceq_\theta B$. ∘

In particular, if the preconditions are fulfilled, all outcomes are possible.

**Example 9.7** As an illustration, consider

$$\begin{array}{ll} \mathtt{on(X,Y)}, \mathtt{cl(X)}, \mathtt{cl(Z)}, & \xleftarrow{0.9:\mathtt{move(X,Y,Z)}} \quad \mathtt{cl(X)}, \mathtt{cl(Y)}, \mathtt{on(X,Z)}, \\ \mathtt{X} \neq \mathtt{Y}, \mathtt{Y} \neq \mathtt{Z}, \mathtt{X} \neq \mathtt{Z} & \qquad\qquad\quad \mathtt{X} \neq \mathtt{Y}, \mathtt{Y} \neq \mathtt{Z}, \mathtt{X} \neq \mathtt{Z} \\ \mathtt{cl(X)}, \mathtt{cl(Y)}, \mathtt{on(X,Z)}, & \xleftarrow{0.1:\mathtt{move(X,Y,Z)}} \quad \mathtt{cl(X)}, \mathtt{cl(Y)}, \mathtt{on(X,Z)}, \\ \mathtt{X} \neq \mathtt{Y}, \mathtt{Y} \neq \mathtt{Z}, \mathtt{X} \neq \mathtt{Z}. & \qquad\qquad\quad \mathtt{X} \neq \mathtt{Y}, \mathtt{Y} \neq \mathtt{Z}, \mathtt{X} \neq \mathtt{Z}. \end{array} \tag{9.4}$$

which moves block $\mathtt{X}$ on $\mathtt{Y}$ with probability 0.9. With probability 0.1 the action fails, i.e., the state does not change. Applied to the above state $z$ the action tells us that $\mathtt{move(a,b,c)}$ will result in $z' \equiv \mathtt{cl(a)}, \mathtt{on(a,b)}, \mathtt{on(b,c)}$ with probability 0.9 and with probability 0.1 we stay in $z$. ∘

[37] For the sake of simplicity, we consider cost-free actions. The framework can be adapted to the case of action costs. Note also that the meaning of *abstract action* here differs from that sometimes used in the context of hierarchical RL, see e.g. [Dietterich, 2000].

This type of action definition implements a kind of probabilistic STRIPS operator [Hanks and McDermott, 1994].

The model $\mathbf{R}$ of abstract rewards specifies the rewards generated by entering abstract states. In our framework, it will also coincide with our initial *abstract state value* function $V_0$.

**Definition 9.8** An *abstract state value function* $V$ is a finite list of *value rules* of the form $c \leftarrow B$ were $B$ is an abstract state and $c \in \mathbb{R}$. ∘

Abstract state value functions typically consist of a set of multiple values rules. They are applied to ground states. In order to avoid conflicting value rules for a single ground state, we apply a similar conflict resolution technique as for logical HMMs, cf. Section 5.1. We assume the value rules to be totally ordered and do assign the first matching value rule such as in Prolog [38]

**Example 9.9** As an illustration, consider $\mathbf{R} = V_0$ with $10.0 \leftarrow \texttt{on}(\texttt{a},\texttt{b})$ and $0.0 \leftarrow \texttt{true}$. It assigns 0 to $z$ but 10 to $z'$. Using `true` in the last value rule assures that all states are assigned a value. ∘

We will also employ *abstract action-state value* functions, which are similar to abstract state value functions and of which an example can be found in Section 10.4.2.

**Definition 9.10** An *abstract state action value* function $Q$ is a finite set of *Q-rules* of the form $c : A \leftarrow B$ were $B$ is an abstract state, $A$ is an action and $c \in \mathbb{R}$. ∘

We apply the same conflict resolution technique as for abstract state value functions.

To summarize, rewards are specified over queries, i.e., existentially quantified goals. Although these are simple, they are expressive enough to specify many interesting problems studied by the (relational) RL community such as *shortest-path problems*. Here, the goal is to reach certain (abstract) states. When a goal state is entered, the process ends. In RL, episodic tasks are encoded using *absorbing states*. We encode them by *artificial* deterministic rules of the form

$$\texttt{absorbing} \leftarrow \texttt{on}(\texttt{a},\texttt{b}),$$

which denotes that all states that are subsumed by $\texttt{on}(\texttt{a},\texttt{b})$ transition only to themselves and generate only zero rewards. For example, $z$ is not absorbing but $z'$ is.

Finally, we need a way to enforce the **integrity constraints** imposed by our domain. For instance, in the `move` definition above we employed symmetry of $\neq$. This can be modeled by a set $\mathbf{C}$ of integrity constraints. Each integrity constraint is a Horn clause.

**Example 9.11** In the blocks world, no block can be free if there is a block on top of it and no block can be on itself. This can be expressed using the constraints $\texttt{false} \leftarrow \texttt{on}(\texttt{X},\texttt{Y}), \texttt{cl}(\texttt{Y})$ and $\texttt{X} \neq \texttt{Y} \leftarrow \texttt{on}(\texttt{X},\texttt{Y})$. ∘

[38] In [Kersting et al., 2004], we proposed to assign the maximal value $c$ of all matching value rules $c \leftarrow B$ to a state as value. This corresponds to totally ordering the rules according to decreasing values.

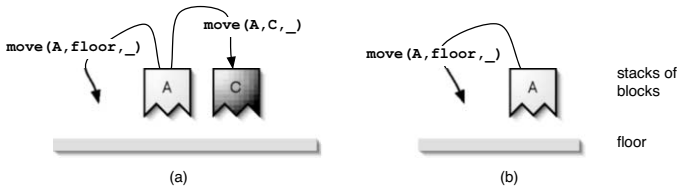



**Figure 9.1.** The two underlying patterns of the blocks world for stacking. (a) There are at least two stacks of height $> 0$. (b) There is only one stack left (the goal state). The serrated cuts indicate that `A` (resp. `C`) can be on top of some other block or on the floor, which is denoted by the anonymous variable `_`.

The *completion* of an abstract state $Z$ is the least fixpoint of $\mathbf{C} \cup \{Z\}$, i.e., all facts deducible from $\mathbf{C} \cup \{Z\}$.

**Example 9.12** The state $\texttt{on}(\texttt{a}, \texttt{b})$ does not encode that `a` is not `b`. Using the rules above, this state is completed to $\texttt{on}(\texttt{a}, \texttt{b}), \texttt{a} \neq \texttt{b}$. ∘

Furthermore, if the completion includes `false`, the state does not satisfy the constraints, i.e., it is an *illegal* state. To deal with integrity constraints, we also have to adapt our notations of action definitions and generality. Action definitions are now constrained so that they cannot lead to illegal states. Furthermore, we employ the integrity constraints as a background theory and use Buntine's *generalized subsumption* framework [1988] to test for subsumption.

By now we are able to formally define logical Markov decision programs.

**Definition 9.13** (Markov Decision Program) A Markov decision program [39] is a tuple $\mathbf{M} = (\Sigma, \mathbf{A}, \mathbf{R}, \mathbf{C})$ where $\Sigma$ is a first-order alphabet, $\mathbf{A}$ is a set of abstract actions, $\mathbf{R}$ is an abstract reward model, and $\mathbf{C}$ is a set of integrity constraints. ∘

Before giving the semantics of Markov decision programs, let us further illustrate Markov decision programs.

**Example 9.14** The goal of the *stack* task in the blocks world is to move all blocks on one single stack. For the sake of simplicity, we again assume that all variables denote different objects. The move action and integrity constraints remain unchanged. The abstract reward model, however, changes to

$$0.0 \leftarrow \texttt{cl}(\texttt{X}), \texttt{cl}(\texttt{Y}), \texttt{on}(\texttt{X}, \texttt{U}), \texttt{on}(\texttt{Y}, \texttt{V}).$$
$$10.0 \leftarrow \texttt{true}.$$

This reward model is governed by the underlying patterns of the blocks world, which are shown in Figure 9.1. Two abstract states (the artificial `absorbing` state excluded)

[39] In [Kersting and De Raedt, 2003, 2004], we also used the term *logical Markov decision processes* (LOMDPs). In this thesis, we use the term *Markov decision programs* to stress their logic programming character.

together with the order in which they occur cover all possible state action patterns in the blocks world. Furthermore, *stack* is an episodic task, i.e., it ends when reaching the goal state. Therefore, we use $\texttt{absorbing} \leftarrow \texttt{cl(X)}, \texttt{not(cl(Y)}, \texttt{X} \neq \texttt{Y})$. Similar, we can easily encode the *unstack* goal:

$$0.0 \leftarrow \texttt{cl(X)}, \texttt{on(X, Y)}, \texttt{on(Y, Z)}.$$
$$10.0 \leftarrow \texttt{true}.$$

with the absorbing state $\texttt{absorbing} \leftarrow \texttt{not(on(X, Y)}, \texttt{on(Y, Z))}$. ∘

Note that we have not specified the number of blocks. The Markov decision program represents all possible blocks worlds using only 6 abstract transitions, i.e. 12 probability and reward parameters, whereas the number of parameters of a propositional system explodes, e.g., for 10 blocks there are $58,941,091$ states.

## 9.3 Semantics

In principle, a Markov decision program $\mathbf{M}$ induces a Markov decision process $M$.

To see this, assume a countably finite set of constants. Let $\mathrm{hb}^a_\Sigma \subset \mathrm{hb}_\Sigma$ be the set of all ground atoms built over abstract **a**ction names, and let $\mathrm{hb}^s_\Sigma \subset \mathrm{hb}_\Sigma$ be the set of all ground atoms built over non-action names. Now, construct $M$ from $\mathbf{M}$ as follows. The countable state set $S$ consists of all finite subsets of $\mathrm{hb}^s_\Sigma$. The set of actions $A(Z) \subset \mathrm{hb}^a_\Sigma$ for state $Z \in S$ consists of all actions applicable in $Z$, i.e., those actions where the preconditions are fulfilled. We have that $|\mathbf{A}(Z)| < \infty$ holds because $Z$ is finite. The probability $P(Z, a, Z')$ of a transition in $\mathbf{T}$ from $Z$ to another state $Z'$ after performing an action $a$ is the probability $p$ associated to the unique abstract transition matching $Z$, $a$, and $Z'$ normalized by the number of transitions of the form $Z'' \overset{a}{\leftarrow} Z$ in $\mathbf{T}$. If there is no abstract transition connecting $Z$ and $Z'$, the probability is zero. The bounded rewards $R(Z, a, Z')$ are constructed in a similar way but are not normalized.

Thus, assuming a finite transition model for each ground state $Z \in \mathrm{hb}_s(\Sigma)$, i.e., $|\{Z' | P(Z, a, Z') > 0\}| < \infty$ the induced MDP $M$ is discrete and from Theorem 6.2.5 in [Puterman, 1994] it follows:

**Corollary 9.15** A Markov decision program $\mathbf{M} = (\Sigma, \mathbf{A}, \mathbf{R}, \mathbf{C})$ specifies a discrete Markov decision process $M = (S, A, \mathbf{T})$, for which an optimal policy exists, if *(*) the transition model for each ground state $Z \in \mathrm{hb}_s(\Sigma)$ is finite.* ∘

In other words, there exists an optimal policy (over ground states) for every Markov decision program fullfilling (*). Interesting cases are functor-free programs, in which no functors occur in abstract transitions, and range-restricted programs (abstract transitions $H_i \overset{p_i:A}{\leftarrow\!\!-} B$ fullfill $\mathrm{vars}(H_i) \subset \mathrm{vars}(B)$) with a finite set of possible starting states. Furthermore, Markov decision programs generalize finite (state set) Markov decision processes, which are typically investigated within reinforcement learning because every finite Markov decision process is a propositional Markov decision program, in which all relation symbols have arity 0.

Markov decision programs, are strictly more expressive than finite Markov decision Processes as the blocks world tasks for any number of blocks show. This is not due to the use of negation but due to logical abstraction and unification.

**Example 9.16** Typical blocks world tasks such as $\texttt{cl}(\texttt{a})$ and $\texttt{on}(\texttt{a},\texttt{b})$ do not require negation to encode their reward models:

$$10.0 \leftarrow \texttt{cl}(\texttt{a}). \quad \text{respectively} \quad 10.0 \leftarrow \texttt{on}(\texttt{a},\texttt{b}).$$
$$0.0 \leftarrow \texttt{true}.$$

We avoid the use of negation by sorting the rules into decreasing order of values. ◦

We will call such Markov decision programs *decreasingly ordered, negation-free* Markov decision programs. They are defined as follows:

**Definition 9.17** (Decreasingly ordered, negation-free Markov decision programs) In decreasingly ordered, negation-free Markov decision programs, value rules are ordered so that their values decrease, and negation is not used to define value rules, absorbing states, and decision rules. ◦

Still, decreasingly ordered, negation-free Markov decision programs generalize finite Markov decision processes because any Markov decision process over a finite set of states is a decreasingly ordered, negation-free Markov decision program. There are only finitely many rewards, which can be ordered appropriately. Furthermore, because there are only finitely many states, transitions can be specified without negation. Therefore, blocks world tasks such as *unstack* and *stack* for finitely many blocks [40] can be encoded as decreasingly ordered, negation-free Markov decision programs. Moreover, tailored exact solution techniques can be developed as we will show in Section 10.4. For the general case, this is still an open question. In the following section, however, we will develop a model-free technique for computing approximative solutions of general Markov decision processes and prove its convergence.

§ 10

# Solving Markov Decision Programs *

*... in which abstract policies are defined, generalized relational policy iteration (GRPI) as a general scheme for learning abstract policies is presented, and two GRPI approaches are introduced, experimentally evaluated and used to establish convergence results for relational reinforcement learning ...*

A solution of a Markov decision process is a policy, mapping states to actions, that determines states transitions that maximize the expected future reward. As Corollary 9.15 states, each Markov decision program $\mathbf{M}$ specifies a discrete Markov decision

[40] Such blocks worlds have been considered in the probabilistic planning track of the international planning competition 2004.

* Builds on [Kersting and De Raedt, 2003, 2004, Kersting et al., 2004].

process $M$, which is solvable. Because a solution for $M$ should also be a solution for $\mathbf{M}$, the existence of an optimal policy $\pi$ for $\mathbf{M}$ is guaranteed: $\pi$ is simply the optimal policy of $M$. Indeed, this policy is extensional or propositional in the sense that it specifies for each ground state separately which action to execute. In turn, specifying such policies for Markov decision programs with large state spaces is cumbersome and learning them will require much effort. This motivates the introduction of *abstract policies*.

## 10.1 Abstract Policies

Many value functions have a rich internal structure. Reconsider the learning agent from the introduction.

**Example 10.1** The agent must learn to use its hand to remove blocks on some block a. The optimal value function would assign the same value, say 8.1, to all states where there is some block on top of a such as $\mathtt{cl(c)}, \mathtt{on(c,a)}$ or $\mathtt{cl(d)}, \mathtt{on(d,a)}, \mathtt{cl(b)}, \mathtt{on(b,e)}$. Thus, we can compress the propositional value function by using value rules such as $8.1 \leftarrow \mathtt{cl(X)}, \mathtt{on(X,a)}$. The same holds for policies. Instead of specifying for each state separately the optimal action, we compress them into $\mathtt{move(X,floor,a)} \leftarrow \mathtt{cl(X)}, \mathtt{on(X,a)}$. ∘

More formally, abstract policies $\pi$ intentionally specify the action to take for sets of states, i.e., for an abstract state.

**Definition 10.2** An *abstract policy* $\pi$ *over* $\Sigma$ *is a finite set of decision rules of the form* $\mathtt{a} \leftarrow \mathtt{L}$*, where* a *is an abstract action and* L *is an abstract state. We assume* a *to be applicable in* L*, i.e.,* $\mathrm{vars}(a) \subseteq vars(L)$ . ∘

The meaning of a single decision rule $a \leftarrow L$ is as follows:

**Semantics 10.3** (Decision Rule) If the agent is in a state $Z$ such that $a \leq_\theta L$, then the agent performs action $a\theta$ with probability $1/|\theta|$, i.e., uniformly with respect to number of possible instantiations of action $a$ in $Z$. ∘

Usually, however, $\pi$ consists of multiple decision rules. We apply the same conflict resolution technique as for abstract state value functions. This means we assume a total order $\prec^\pi$ among the decision rules in $\pi$ and use the first matching decision rule such as in Prolog.

**Example 10.4** Consider the following *unstack-stack* abstract policy:

$$
\begin{aligned}
\langle 1\rangle\ & \mathtt{move(A,floor,B)} \leftarrow \mathtt{on(A,B)}, \mathtt{on(C,D)}, \mathtt{on(E,floor)}, \mathtt{cl(A)}, \mathtt{cl(C)}, \mathtt{cl(E)}.\\
\langle 2\rangle\ & \mathtt{move(A,floor,B)} \leftarrow \mathtt{on(A,B)}, \mathtt{on(C,D)}, \mathtt{cl(A)}, \mathtt{cl(C)}.\\
\langle 3\rangle\ & \mathtt{move(E,A,floor)} \leftarrow \mathtt{on(A,B)}, \mathtt{on(E,floor)}, \mathtt{cl(A)}, \mathtt{cl(E)}.\\
\langle 4\rangle\ & \mathtt{move(A,B,floor)} \leftarrow \mathtt{cl(A)}, \mathtt{cl(B)}.\\
\langle 5\rangle\ & \mathtt{stop} \leftarrow \mathtt{on(A,B)}, \mathtt{cl(A)}.
\end{aligned}
$$

where the start action adds the absorbing propositions, i.e., it encodes that we enter an absorbing state. We have omitted the absorbing state in front and statements that

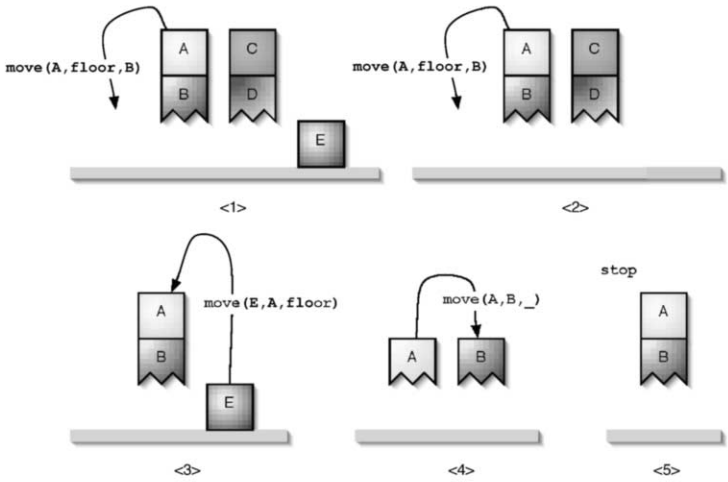



**Figure 10.1.** The decision rules of the *unstack-stack* policy. In the figure, the decision rules are ordered from left to right, i.e., a rule fires only if no rule further to the left fires.

variables refer to different blocks. For instance in state $z$ (see before), only decision rule $\langle 3 \rangle$ fires.

The policy, which is graphically depicted in Figure 10.1 and is interesting for several reasons. First, it is close to the *unstack-stack* strategy, which is well known in the planning community [Slaney and Thiébaux, 2001]. Basically, the strategy amounts to first putting all blocks on the table and then building the goal state by stacking all blocks from the floor onto one single stack. No block is moved more than twice. Second, it perfectly generalizes to all other blocks worlds, no matter how many blocks there are. Finally, it cannot be learned in a propositional setting because here the optimal, propositional policy would encode the number of states and the optimal number of moves. ○

Due to the conflict resolution, abstract policies induce a partition of the state space and, hence, their semantics can be stated in terms of (ground) policies.

**Proposition 10.5** *Any abstract policy $\pi$ specifies a* nondeterministic *policy $\pi$ at the level of ground states. A (stationary) nondeterministic policy $\pi$ maps a state to a distribution over actions.* ○

*Proof* Let $\mathcal{L} = \{L_1, \ldots, L_m\}$ be the set of bodies in $\pi$ (ordered with respect to $\prec^\pi$). We call $\mathcal{L}$ the *abstraction level* of $\pi$ and assume that it covers all possible states of the Markov decision program. This together with the total order guarantees that $\mathcal{L}$ forms a partition of the states. The equivalence classes $[L_1], \ldots, [L_m]$ induced by $\mathcal{L}$ are inductively defined by $[L_1] = S(L_1)$ and $[L_i] = S(L_i) \setminus \bigcup_{j=1}^{i-1} [L_j]$ for $i \geq 2$, where $S(L_i)$ denote the set of states covered by the abstract state $L_i$. Because we choose uniformly among all instances of an action, the proposition holds. □

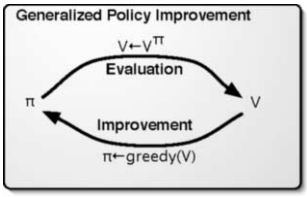



**Figure 10.2.** Generalized policy iteration as illustrated in [Sutton and Barto, 1998]. Optimal policies are computed by two interacting processes. Policy evaluation makes the value function consistent with the current policy, and policy improvement makes the policy greedy with respect to the current value function. *greedy* denotes the greedy policy computed from the current value function.

The crucial question for Markov decision programs and for relational reinforcement learning is therefore, how one can learn abstract policies?

## 10.2 Generalized Relational Policy Iteration

According to Sutton and Barto [1998], almost all MDP solvers and reinforcement learning systems follow the so called *generalized policy iteration* (GPI) scheme shown in Figure 10.2. It consists of two interacting processes: *policy evaluation* and *policy improvement*. Here, evaluating a policy refers to computing the value function of the current policy, and policy improvement refers to computing a new policy based on the current value function. For instance, the *greedy policy* with respect to the state action value function is $\pi(s) = \arg\max_a Q^\pi(s,a)$.

Indeed, GPI assumes a fixed level of abstraction. Consequently, it cannot directly be applied to learn abstract policies because — similar to learning the structure of logical HMMs, see Section 7 — learning abstract policies requires exploring different abstraction levels.

**Example 10.6** Reconsider the *unstack-stack* abstract policy from Example 10.4. To eventually learn this policy from scratch, several abstract policies have to be explored such as the policy which has the additional decision rule

$$\langle 0 \rangle \ \mathtt{move(A,floor,B)} \leftarrow \mathtt{on(A,B)}, \mathtt{on(C,D)}, \mathtt{on(D,floor)}, \mathtt{on(E,floor)}, \mathtt{cl(A)}, \mathtt{cl(C)}, \mathtt{cl(E)}.$$

or the policy where decision rule $\langle 1 \rangle$ is missing. ∘

Therefore, we interleave GPI with the additional process of *policy refinement*, which makes small modifications to an abstract policy. Thus, the resulting *generalized relational policy improvement* (GRPI) scheme iteratively modifies the current abstract policy syntactically, evaluates it, and improves it, as illustrated in Figure 10.3. To modify an abstract policy, one can apply ILP techniques such as refinement operators, which we reviewed in Section 2.2.

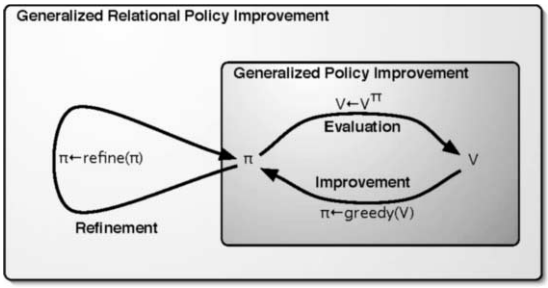



**Figure 10.3.** Generalized relational policy iteration, which accounts for different abstraction levels. It is an upgrade of generalized policy iteration for traditional reinforcement learning as in Figure 10.2. *greedy* denotes the greedy policy computed from the current value function, see [Sutton and Barto, 1998].

**Example 10.7** Continuing our last example, the *unstack-stack* policy may be generalized by deleting decision rule $\langle 1 \rangle$. It may also be refined by adding the specialized variant $\langle 0 \rangle$ of decision rule $\langle 1 \rangle$. ∘

One can, however, do even better.

When a model of the environment's dynamics, i.e., a Markov decision program is known, it can be used to score different refinements (e.g., measuring the *influence* of a refinement of one state on the remaining states [Munos and Moore, 1999, Kim and Dean, 2003]) or to model-driven compute the best level of abstraction directly. We will develop such a model-driven GRPI approach in Section 10.4 when introducing the relational Bellman update operator ReBel. ReBel uses Bellman's update operator (9.2) to evaluate abstract policies and the *greatest-lower-bound* and $\theta$-subsumption for clauses, see Section 2.2, to refine abstract policies.

In the model-free case, i.e., when no Markov decision program is given, the agent can employ the experience she already has, i.e., the states visited to compute and evaluate different abstract levels. This approach is followed by Džeroski et al. [2001] within RRL-RT. RRL-RT interleaves the policy evaluation and policy refinement processes. Starting from some initial abstraction level, RRL-RT induces a logical regression tree [Blockeel and De Raedt, 1998] from episodes/traces of states (i.e., interpretations) where the states are weighted by their currently expected rewards. The learned logical regression tree is an approximation of the abstract state(-action) value function and is used to compute a new policy (policy improvement). Based on this new policy, RRL-RT generates new episodes to improve its current approximation of the value function. RRL-RT iterates these steps until convergence. Thus, RRL-RT can be viewed as an instance of GRPI.

Empirically, RRL-RT has been shown to work well on a wide range of domains such as blocks world, Tetris, and Digger. In the next Section, we will provide a first step in explaining why it works well. More precisely, we will focus on the *relational evaluation problem*, see Definition 10.8 in the next section, within GRPI approaches and prove convergence for a relational variant of TD($\lambda$). The learning setting employed is that

of *probabilistic learning from proofs* when we view sequences as proofs. One can also use the *probabilistic learning from interpretations* setting because the agent observes sequences of interpretations. The convergence result presented is, to the best of our knowledge, the first in the context of relational reinforcement learning. Thus, we will make a first step into the direction of a theory of relational reinforcement learning.

## 10.3 Model-free Relational TD($\lambda$)

The *relational evaluation problem* considers how to compute the state-value function $V^\pi$ for an arbitrary abstract policy $\pi$. In this Section, we will focus on model-free approaches. Model-free approaches do not know the reward and the transition functions in advance when computing the value of an abstract policy from experiences.

**Definition 10.8** (Relational Policy Evaluation Problem) **Given** an abstract policy $\pi$, **find** the state-value function $V^\pi$ from experiences $\langle S_t, a_t, S_{t+1}, r_t \rangle$ only, where action $a_t$ leads from state $S_t$ to state $S_{t+1}$ receiving reward $r_t$. ∘

In contrast to traditional model-free approaches, maintaining values for all states of the underlying MDP $M$ is not feasible.

### 10.3.1 The Algorithm

The basic idea to come up with a relational evaluation approach is to define the expected reward of a state $L_i$ of an abstraction level $\mathcal{L} = \{L_1, \ldots, L_m\}$ to be the average expected value for all the states in $[L_i]$. This is a good model because if we examine each state in $[L_i]$, we make contradictory observations of rewards and transition probabilities. The best model is the average of these observations given no prior knowledge of the model. To prove convergence [41], we reduce the "abstract" evaluation problem to the evaluation problem for the underlying MDP $M = (S, A, \mathbf{T}, \lambda)$ with state aggregation (see e.g. [Gordon, 1996, Singh et al., 1995, Kim and Dean, 2003]) with respect to $[L_1], \ldots, [L_m]$. For ease of explanation, we will focus on a $TD(0)$ approach [42], see e.g. [Sutton and Barto, 1998]. Results for general $TD(\lambda)$ can be obtained by applying Tsitsiklis and Van Roy's [1997] results.

Algorithm III.1 sketches *relational* $TD(0)$. Given some experience following an abstract policy $\pi$, RTD(0) updates its estimate $\widehat{V}$ of $V$. If the estimated is not changing considerably, cf. line 12, the algorithm stops. If an absorbing state is reached, cf. line 11, If a nonabsorbing state is visited, cf. line 4, then it updates its estimate, cf. line 9, based on what happens after that visit, cf. lines 6–8. Instead of updating the estimate at the level of states, $RTD(0)$ updates its estimate at the abstraction level $\mathcal{L}$ of $\pi$.

[41] It has been experimentally shown that (already model-based) reinforcement learning with function approximation does not converge in general, see e.g. [Boyan and Moore, 1995]. Fortunately, this does not hold for averagers, which we employ here.

[42] A similar analysis can be done for model-based approaches. Gordon [1996] showed that value iteration with an averager as function approximator converges within a bounded distance from the optimal value function of the original MDP.

**Algorithm III.1**: Relational TD(0) where $\alpha$ is the learning rate and $\widehat{V}(\mathcal{L})$ is the approximation of $V(\mathcal{L})$.

```
1  Let π be an abstract policy with abstraction level L
2  Initialize V̂_0(L) arbitrarily for each L in L
3  repeat
4      Pick a ground state Z of the underlying MDP M
5      repeat
6          Choose action a in Z based on π as described in Section 10.1, i.e., (1) select
           first decision rule a ← L in π that matches according to ≺^π, (2) select aθ
           uniformally among induced ground actions
7          Take aθ, observe r and successor state Z' as described in Section 9.2, i.e.,
           (1) select with probability p_i the i-th outcome of aθ, (2) compute Z' as
           [b \ Bθ] ∪ H_iθ
8          Let L' in L be the abstract state first matching Z' according to ≺^π
9          V̂(L) := V̂(L) + α · (r + λ · V̂(L') − V̂(L))
10         Set Z := Z'
11     until Z is terminal, i.e., absorbing
12 until converged or some maximal number of episodes exceeded
```

### 10.3.2 Proof of Convergence

To show convergence, it is sufficient to reduce *RTD*(0) to *TD*(0) with soft state aggregation [Singh et al., 1995]. The basic idea of soft state aggregation is to cluster the state space, i.e., to map the state space $S$ into clusters $c_1, \ldots, c_k$. Each state $s$ belongs to a cluster $c_i$ with a certain probability $P(c_i|s)$. The value function then is computed at the level of clusters rather than states. Relational TD(0) is a special case of soft state aggregation. To see this recall that the abstraction level $\mathcal{L}$ partitions the state space $S$. Thus, one can view the abstract states in $\mathcal{L}$ as clusters where each state $Z \in S$ belongs to only one cluster $[L_i]$, i.e., $P([L_i] \mid Z) = 1$ if $Z \in S(L_i)$; otherwise $P([L_i] \mid Z) = 0$. Furthermore, the state set $S$ and the action set $A$ of the underlying MDP $M$ are finite, and the agent follows a nondeterministic policy. Therefore, the assumptions of the following Theorem are fulfilled.

**Theorem 10.9** *TD*(0) *with soft state aggregation applied to* $M$ *while following an abstract policy* $\pi$ *converges with probability one to the solution of the following system of equations:*

$$V([L_i]) = \sum_{Z \in S} P^\pi(Z \mid [L_i]) \left[ R^\pi(Z) + \lambda \sum_{L_j \in \mathcal{L}} P^\pi(Z, [L]_j) V([L_j]) \right] \tag{10.1}$$

*for all* $L_i$ *in* $\mathcal{L}$. ∘

*Proof sketch:* The theorem is a direct reformulation of Corollary 2 in [Singh et al., 1995] in terms of the partition induced by the abstract policy $\pi$. □

From Theorem 10.9, it follows that *RTD*(0) applied to a Markov decision program $\mathcal{M}$ while following an abstract policy $\pi$ at abstraction level $\mathcal{L}$ converges with probability one to the solutions of the system of equations (10.1).

Note that for arbitrary abstraction levels, despite that Theorem 10.9 shows that *RTD*(0) learning will find solutions, the error in the (ground) state space will not be zero in general. Equation (10.1) basically states that an abstract policy $\pi$ induces a process $\mathbf{L}$ over $[L_1], \ldots, [L_m]$ whose transition probabilities and rewards for a state $[L_i]$ are averages of the corresponding values of the covered ground states in $M$, see also [Kersting and De Raedt, 2003]. Therefore, the process $\mathbf{L}$ appears to a learner to have a non-Markovian nature.

**Example 10.10** Consider the following Markov decision program with actions

$$1\text{: } \mathtt{q} \xleftarrow{1.0:a1} \mathtt{p}, \mathtt{q} \qquad 2\text{: } \emptyset \xleftarrow{1.0:a2} \mathtt{p} \qquad \text{and} \qquad 3\text{: } \mathtt{p} \xleftarrow{1.0:a3} \mathtt{true},$$

abstract reward model

$$0.0 \leftarrow \mathtt{p}, \mathtt{q}$$
$$1.0 \leftarrow \mathtt{p}$$
$$0.0 \leftarrow \mathtt{true}$$

and the abstraction level $\mathcal{L} = \{\mathtt{p}, \mathtt{q}, \mathtt{true}\}$ . Here, the values for [q] and [true] are the same in $\mathbf{L}$ as the next state is the same, namely [p]. The underlying MDP $M$, however, assigns different values to both as the following traces show:

$$\mathtt{q} \xrightarrow{1.0:a3} \mathtt{p}, \mathtt{q}, \ \mathbf{0.0} \xrightarrow{1.0:a1} \mathtt{q}, \ \mathbf{0.0} \ldots \quad \text{and} \quad \mathtt{true} \xrightarrow{1.0:a3} \mathtt{p}, \ \mathbf{1.0} \xrightarrow{1.0:a2} \mathtt{true}, \ \mathbf{0.0} \ldots$$

where the bold numbers denote the rewards. ∘

Nevertheless, *RTD*(0) converges at the level of $\mathcal{L}$ and can generalize well even for unseen ground states due to the abstraction.

To summarize, Theorem 10.9 shows that temporal-difference evaluation of an abstract policy converges. Different policies, however, will have different errors in the ground state space. In the context of GRPI, Theorem 10.9 suggests to use refine heuristically the abstraction level to reduce the error in the ground state space. This is akin to the approach taken in RRL-RT, which reads as follows:

(1) Because each logical regression tree induces a finite abstraction level, temporal-difference evaluation of a fixed regression tree converges.
(2) Relational node/state splitting (based on the state sequences encountered so far) is used to heuristically reduce the error in the ground state space.

Theorem 10.9, however, assumes that each state is infinitely often visited. This is of course an unrealistic assumption. To investigate how well relational TD(0) performs in practice, we conducted a series of experiments.

### 10.3.3 Experimental Evaluation

The experiments put the following hypotheses to test:

**H1** RTD(0) can converge in finite time for finite abstraction levels in practice
**H2** Using *RTD*(0), abstract policies can be compared.
**H3** RTD(0) works for actions with multiple outcomes.
**H4** Relational policy refinement is needed.
**H5** Variance can be used as a heuristic state-splitting criterion.

Our task was to evaluate abstract policies within the blocks world. In contrast to the similar experiments reported by Džeroski et al. [2001] on RRL-RT, we exclusively use the standard predicates `on`, `cl`, and `bl`. Džeroski et al. also needed to make use of several background knowledge predicates such as `above`, `height` of stacks as well as several directives to the first-order regression tree learner. Another difference to our approach is that RRL-RT induces the relevant abstract states automatically using a regression tree learner. Our goal, however, was not to present an overall GRPI system.

We implemented *RTD*(0) using the Prolog system YAP-4.4.4. All experiments were run on a 3.1 GHz Linux machine and with a discount factor $\lambda$ of 0.9 and a learning rate $\alpha$ of 0.015. We randomly generated 100 blocks world states for 6 blocks, for 8 blocks, and for 10 blocks using the procedure described by Slaney and Thiébaux [2001]. This set of 300 states constituted the set *Start* of starting states in all experiments. Note that for 10 blocks a traditional MDP would have to represent $58,941,091$ states of which $3,628,800$ are goal states. The result of each experiment is an average of five runs of 5000 episodes, where for each new episode we randomly selected one state from *Start* as starting state. For each run, the value function was initialized to zero. Note that in all experiments, the abstract policies and value functions apply no matter how many blocks there are.

**Experiment 1:** Our task was to evaluate the *unstack-stack* abstract policy for the *stack* Markov decision program introduced above. The results are summarized in Figure 10.4 **(a)** and clearly show that hypothesis **H1** holds. The learning curves show that the values of the abstract states converged and, hence, *RTD*(0) converged. Note that the value of abstract state $\langle 5 \rangle$ remained 0. The reason for this is that, by accident, no state with all blocks on the floor was in *Start*. Furthermore, the values converged to similar values in all runs. The values basically reflect the nature of the policy. It is better to have a single stack than multiple ones. The total running time for all 25000 episodes was 67.5 seconds measured using YAP's build-in `statistics(runtime,_)`.

**Experiment 2:** In reinforcement learning, *policy improvement* refers to computing a new policy based on the current value function. In a relational setting, the success of, e.g., computing the greedy policy given an abstract value function depends on the granularity of the value function. For instance, based on the last value function, it is not possible to distinguish between `move(A,floor,_)` and `move(A,B,_)` as actions in decision rule $\langle 1 \rangle$ because both would get the same expected values. To overcome this, one might refine the abstraction level (see experiment 5) or evaluate different policies at the same abstraction level. In this experiments, we evaluated a

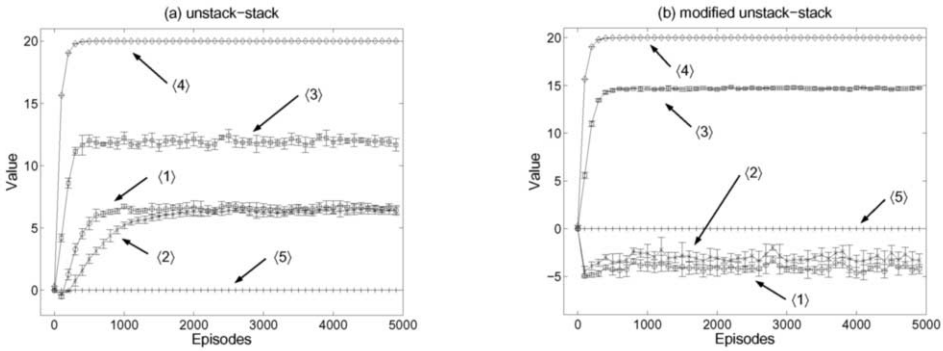



**Figure 10.4.** Relational TD(0)'s learning curves on the evaluation problem **(a)** for the *unstack-stack* policy and **(b)** for the *modified* unstack-stack policy. The predicted values are shown as a function of the number of episodes. These data are averages over 5 runs; the error bars show the standard deviations.

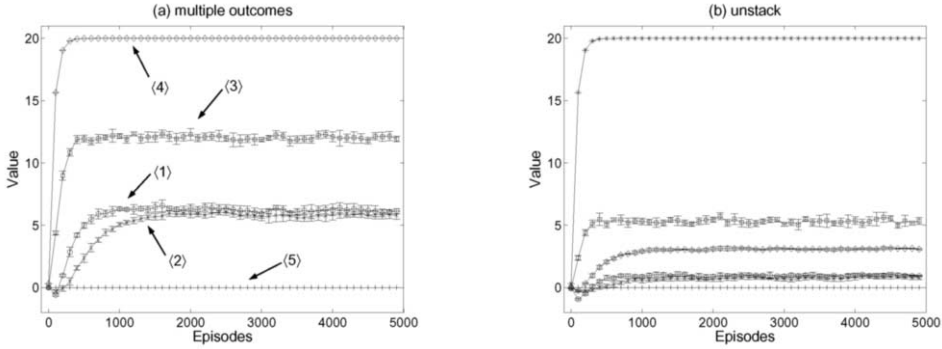



**Figure 10.5.** Relational TD(0)'s learning curves on the evaluation problem for the *unstack-stack* policy where **(a)** the actions of the underlying Markov decision program have multiple outcomes and **(b)** the underlying Markov decision program encoded the *unstack* problem. The predicted values are shown as a function of the number of episodes. These data are averages over 5 runs; the error bars show the standard deviations.

*modified* 'unstack-stack' policy in the same way as in the first experiment. It differs from the 'unstack-stack' policy in that we do not perform `move(A,floor,B)` but `move(E,A,floor)` in the decision rule $\langle 1 \rangle$. The total running time of all 25,000 episodes increased to 97.3 seconds as the average length of an episode increased. The computed values are summarized in Figure 10.4 **(b)**. Interestingly, the values of abstract states $\langle 1 \rangle$ and $\langle 2 \rangle$ dropped from approximately 5 to approximately $-4$. It is less interesting to pile blocks if there are more than one on piles of height at least 2 left. Furthermore, it becomes more interesting to pile blocks on the only pile of blocks left; the value of $\langle 3 \rangle$ increased. Selecting the decision rules with the highest values from both policies

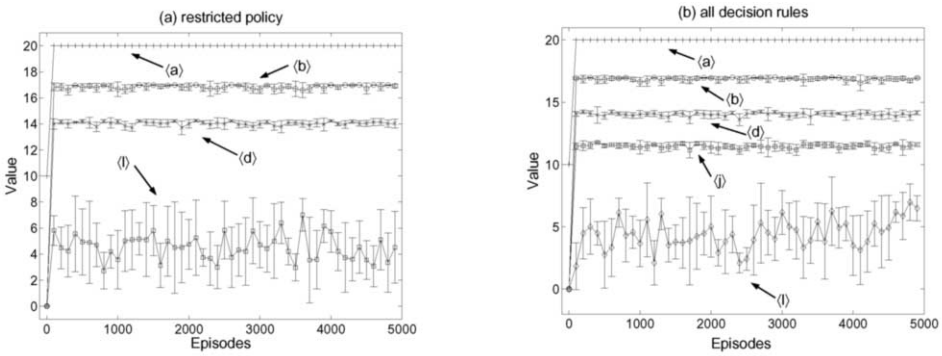



**Figure 10.6.** Learning curves for $RTD(0)$ on the evaluation problem for the `on(a,b)` policy where the underlying Markov decision program encoded the `on(a,b)` goal. The predicted values are shown as a function of the number of episodes. These data are averages over 5 runs; the error bars show the standard deviations; only states with a non zero value are shown (note that we used a finite set of starting states only). **(a)** Policy restricted to $\langle a \rangle - \langle e \rangle$ and $\langle l \rangle$. **(b)** All decision rules $\langle a \rangle - \langle l \rangle$.

results in the original 'unstack-stack' policy. Thus, the *unstack-stack* is preferred over modified one. This shows that hypothesis **H2** holds.

**Experiment 3:** We reran the first experiment where the underlying Markov decision program now encoded that `move` has multiple outcomes. For action $\texttt{move}(\texttt{A}, \texttt{floor}, \texttt{C})$, block `A` may fall on `C`, and for action $\texttt{move}(\texttt{A}, \texttt{C}, \texttt{floor})$ the block `A` may fall on the `floor`. Again, $RTD(0)$ converged as Figure 10.5 **(a)** shows. This shows that hypotheses **H3** holds.

**Experiment 4:** We reran the first experiment, i.e., we evaluated the *unstack-stack* policy, but now the underlying Markov decision program encoded the *unstack* problem. Again, $RTD(0)$ converged as Figure 10.5 **(b)** shows. The running time over all 25000 episodes, however, increased to 317.7 seconds as the underlying Markov decision program was more complex. This shows that hypothesis **H1** holds.

**Experiment 5:** $RTD(0)$ can also be used when there are costs associated with actions. To experimentally validate this, we finally investigated action costs with $\texttt{on}(\texttt{a}, \texttt{b})$ as goal. The underlying Markov decision program was

$$\begin{array}{rcl}
\texttt{absorb} & \xleftarrow{\texttt{1.0:20:stop}} & \texttt{on(a,b)}. \\
\texttt{on(A,floor)}, \texttt{cl(A)}, \texttt{on(C,D)}, \texttt{cl(C)}, \texttt{cl(B)} & \xleftarrow{\texttt{0.9:-1:move(A,floor,B)}} & \texttt{on(A,B)}, \texttt{cl(A)}, \texttt{on(C,D)}, \texttt{cl(C)}. \\
\texttt{on(A,C)}, \texttt{cl(A)}, \texttt{on(C,D)}, \texttt{cl(B)} & \xleftarrow{\texttt{0.9:-1:move(A,C,B)}} & \texttt{on(A,B)}, \texttt{cl(A)}, \texttt{on(C,D)}, \texttt{cl(C)}. \\
\texttt{on(A,floor)}, \texttt{cl(A)}, \texttt{cl(B)} & \xleftarrow{\texttt{1.0:-1:move(A,floor,B)}} & \texttt{on(A,B)}, \texttt{cl(A)}.
\end{array}$$

where the second number on the arrow are the immediate costs of taking an action. Furthermore, we assumed that all variables denote different objects and we omitted `absorb`. If the transition probabilities do not sum to 1.0 then there is an additional abstract transition for staying in the current abstract state. Following the same experimental setup as in the first experiment but using a step size $\alpha = 0.5$, we evaluated

two different policies, namely $\langle a\rangle - \langle e\rangle, \langle l\rangle$ and $\langle a\rangle - \langle l\rangle$ where

$\langle a\rangle$ stop ← on(a, b).
$\langle b\rangle$ move(a, b, B) ← cl(a), cl(b), on(a, B).
$\langle c\rangle$ move(b, floor, a) ← cl(b), on(a, C), on(b, a).
$\langle d\rangle$ move(A, floor, a) ← cl(b), cl(A), on(a, C), on(A, a).
$\langle e\rangle$ move(A, floor, b) ← cl(a), cl(A), on(a, C), on(A, b).
$\langle f\rangle$ move(A, floor, b) ← cl(A), on(a, D), on(b, a), on(A, b).
$\langle g\rangle$ move(b, floor, D) ← cl(b), on(a, C), on(b, D), on(D, a).
$\langle h\rangle$ move(a, floor, C) ← cl(a), on(a, C), on(C, b).
$\langle i\rangle$ move(a, floor, D) ← cl(A), on(a, D), on(A, b).
$\langle j\rangle$ move(A, floor, D) ← cl(b), cl(A), on(a, C), on(A, D), on(D, a).
$\langle k\rangle$ move(A, floor, D) ← cl(a), cl(A), on(a, C), on(A, D), on(D, b).
$\langle l\rangle$ move(A, floor, D) ← cl(A), on(A, D).

In both cases, *RTD*(0) converged as Figure 10.6 shows. The running time over all 25000 episodes was 4.85 seconds ($\langle a\rangle - \langle e\rangle, \langle l\rangle$) and 6.7 seconds ($\langle a\rangle - \langle l\rangle$). In both experiments, state $\langle l\rangle$ was exceptional. It obeyed a higher variance than the other states. The reason is that it acts as a kind of "container" state for all situations, which are not covered by the preceding abstract states. In the refined policy, all added states showed low variances. Thus, we may iterate and refine $\langle l\rangle$ even more. The experiments show that hypotheses **H1**, **H4**, and **H5** hold and supports the variance-based state-splitting approach taken in RRL-RT [Džeroski et al., 2001].

Model-free approaches such as RTD($\lambda$) tackle a difficult problem: estimating value functions without a model of the environment. A model of the environment should yield more efficient solution techniques. In the next section, we will show that this is indeed the case. To the expense of a less expressive language, namely that of decreasingly ordered, negation-free Markov decision programs (see Definition 9.17), we will develop an exact value iteration method.

## 10.4 Model-based Relational Value Iteration based on ReBel

In traditional MDPs and reinforcement learning, as discussed in Section 9.1, the Bellman backup operator (9.2) is one of the central concepts. We will now develop a relational Bellman backup operator, called ReBel, and, based on it, a relational value iteration algorithm. We focus here on decreasingly ordered, negation-free Markov decision programs because their restrictive language allows us to devise ReBel. More expressive language call for more complex solution techniques. For instance, Boutilier et al. [2001] employed the situation calculus; although their work is certainly elegant and principled, due to the complexity of the language, they neither report on a complete implementation nor present automated experiments. Only recently, Sanner and Boutilier [2004, 2005] fully implemented Boutilier et al. approach using more difficult, mathematical techniques.

More formally, we investigate the following problem.

**Definition 10.11** (Model-Based State Value Function Estimation Problem) **Given** a Markov decision program with abstract reward model **R**, i.e., initial abstract state value function $V_0$, **find** the next abstract state value functions $V_t$, $t = 1, 2, \ldots$ ∘

Although, we consider here decreasingly ordered, negation-free Markov decision programs, we will refer to them — for the sake of brevity — as Markov decision programs for the remaining of the section.

The main idea is to upgrade Bellman's traditional backup operator in Equation (9.2). Basically, Equation (9.2) iteratively updates state values based on value approximations of succeeding states. Therefore, we iterate over the following three steps:

(1) Regress all preceding abstract states from $V_t$.
(2) Compute $Q_{t+1}$ over the regressed states based on the value approximations $V_t$ of their succeeding abstracts states.
(3) Compute $V_{t+1}$ by maximizing over $Q_{t+1}$.

Let us now discuss each step in turn in more details.

### 10.4.1 Regression

Let $V_t$ be the current abstract state value function and consider some abstract action, say `move`. For a single Bellman backup, all abstract states $S$, which lead to a condition, i.e., abstract state $S'$ in $V_t$ when taking action `move` have to be computed. Thus, we have to reason from post- to preconditions.

**Example 10.12** Consider $S \equiv \big(\texttt{cl(a)}, \texttt{cl(b)}, \texttt{on(a,c)}, \texttt{on(b,d)}\big)$ (inequality constraints omitted) together with the `move` definition in Equation (9.4). The first outcome of $\texttt{move(a,b,c)}$ can lead from $S$ to the abstract state $S' \equiv \texttt{on(a,b)}$. ∘

Thus, we have to compute the weakest preconditions for the outcomes of an abstract action such as `move` and an abstract state $S'$.

**Definition 10.13** All abstract states $S$, which lead to $S'$ when following some action rule $H_i \xleftarrow{p_i:A} B$, constitute the so called *weakest precondition* $\text{wp}_i(A, S')$ of the $i$-th outcome of $A$. ∘

**Example 10.14** The abstract state $S \equiv \big(\texttt{cl(a)}, \texttt{cl(b)}, \texttt{on(a,c)}, \texttt{on(b,d)}\big)$ from the last example lies in the weakest precondition of $S'$, i.e., $S \in \text{wp}_1(\texttt{move(X,Y,Z)}, S')$ but it does not lie in $\text{wp}_2(\texttt{move(X,Y,Z)}, S')$. ∘

To compute the weakest precondition $\text{wp}_1(\texttt{move(X,Y,Z)}, S')$ we can assume that we "moved" from $S$ to $S'$. Thus,

(1) the preconditions of the action (rule) are fulfilled in $S$, and
(2) $S'$ is partially caused by the first outcome of the action.

**Algorithm III.2**: WeakestPre returns the weakest precondition $\mathrm{wp}_i$ of action rule $H_i \xleftarrow{p_i:A} B$ and abstract state $S'$ given a set of integrity constraints $\mathbf{C}$. We omitted the constraint that only legal and completed abstract states are inserted in $\mathrm{wp}_i$.

```
1  initialize wp_i to be the empty list
2  foreach S'' ⊆ S' and P ⊆ H_i do
3  |  foreach θ = mgu(S'', P) or S'' == ∅ ∧ P == H_i, i.e., θ = ∅ do
4  |  |  S := (S'θ \ Pθ) ∪ Bθ
5  |  |  for all pairs (l, l') ∈ {(l, l') | l ∈ (S'θ \ Pθ) ∧ l' ∈ H_iθ ∪ Bθ} do
6  |  |  |  if mgu(l, l') exists then
7  |  |  |  |  add l ≠ l' to S
8  |  |  |
9  |  |  add all simplifications of S to wp_i
10 |
11 return wp_i
```

As an illustration of 2), reconsider $S' \equiv \mathtt{on}(\mathtt{a},\mathtt{b})$. There are basically two situations. First, **move caused on(a, b)**. That means, we have been in abstract state

$$S_1 \equiv \big(\mathtt{cl}(\mathtt{a}), \mathtt{cl}(\mathtt{b}), \mathtt{on}(\mathtt{a},\mathtt{Z}), \mathtt{a} \neq \mathtt{b}, \mathtt{a} \neq \mathtt{Z}, \mathtt{b} \neq \mathtt{Z}\big)$$

and moved $X = \mathtt{a}$ on $Y = \mathtt{b}$. Second, **move did not cause on(a, b)**. This means, we moved X on Y but not a on b. Therefore, we have been in abstract state

$$T \equiv \big(\mathtt{cl}(\mathtt{X}), \mathtt{cl}(\mathtt{Y}), \mathtt{on}(\mathtt{X},\mathtt{Z}), \mathtt{on}(\mathtt{a},\mathtt{b}), \mathtt{X} \neq \mathtt{Y}, \mathtt{X} \neq \mathtt{Z}, \mathtt{Y} \neq \mathtt{Z}\big),$$

which satisfies that we did not move a on b, i.e., $\mathtt{on}(\mathtt{X},\mathtt{Y}) \neq \mathtt{on}(\mathtt{a},\mathtt{b})$, and that we did not move a from b away, i.e., $\mathtt{on}(\mathtt{X},\mathtt{Z}) \neq \mathtt{on}(\mathtt{a},\mathtt{b})$. These two constraints $C$ guarantee that applying $\mathtt{move}(\mathtt{X},\mathtt{Y},\mathtt{Z})$ in $T$ preserves $\mathtt{on}(\mathtt{a},\mathtt{b})$. The definition of $T \wedge C$ simplifies to $S_2 \equiv \big(T \wedge \mathtt{X} \neq \mathtt{a}\big)$, $S_3 \equiv \big(T \wedge \mathtt{X} \neq \mathtt{a} \wedge \mathtt{Z} \neq \mathtt{b}\big)$, $S_4 \equiv \big(T \wedge \mathtt{Y} \neq \mathtt{b} \wedge \mathtt{X} \neq \mathtt{a}\big)$, and $S_5 \equiv \big(T \wedge \mathtt{Y} \neq \mathtt{b} \wedge \mathtt{Z} \neq \mathtt{b}\big)$. The abstract state $S_2, S_3, S_4$, and $S_5$ are completed (see end of Section 9.2) to the same abstract state, namely

$$S_6 \equiv \big(\mathtt{cl}(\mathtt{A}), \mathtt{cl}(\mathtt{B}), \mathtt{on}(\mathtt{a},\mathtt{b}), \mathtt{on}(\mathtt{A},\mathtt{C})\big)$$

where all variables and constants are mutually different.

Putting everything together, the abstract states $S_1$ and $S_6$ together logically define the weakest precondition of move and $S'$, i.e., $\mathrm{wp}_1(\mathtt{move}(\mathtt{X},\mathtt{Y},\mathtt{Z}), S') \equiv \big(S_1 \vee S_6\big)$.

So far, we considered a single effect only, namely $\mathtt{on}(\mathtt{a},\mathtt{b})$. The method WeakestPre in Algorithm III.2 treats the general case of multiple (combined) effects that are or that are not caused by taking some abstract action such as move. Consider $S' \equiv \big(\mathtt{on}(\mathtt{a},\mathtt{b}), \mathtt{on}(\mathtt{c},\mathtt{d})\big)$. Moving a block on some other block can have caused either $\mathtt{on}(\mathtt{a},\mathtt{b})$ or $\mathtt{on}(\mathtt{c},\mathtt{d})$, or neither of them. Assume that neither of them was caused, i.e., $S''$ is empty and $P = H_1$. The MGU $\theta$ is the empty substitution and $S \equiv \big(\mathtt{on}(\mathtt{a},\mathtt{b}), \mathtt{on}(\mathtt{c},\mathtt{d}), \mathtt{cl}(\mathtt{X}), \mathtt{cl}(\mathtt{Y}), \mathtt{on}(\mathtt{X},\mathtt{Z})\big)$ (inequality constraints omitted) is a possible preimage. Because we know that move did not cause $\mathtt{on}(\mathtt{a},\mathtt{b}), \mathtt{on}(\mathtt{c},\mathtt{d})$, it holds

$$\mathtt{on}(\mathtt{X},\mathtt{Z}) \neq \mathtt{on}(\mathtt{a},\mathtt{b}) \wedge \mathtt{on}(\mathtt{X},\mathtt{Z}) \neq \mathtt{on}(\mathtt{c},\mathtt{d}) \wedge \mathtt{on}(\mathtt{X},\mathtt{Y}) \neq \mathtt{on}(\mathtt{a},\mathtt{b}) \wedge \mathtt{on}(\mathtt{X},\mathtt{Y}) \neq \mathtt{on}(\mathtt{c},\mathtt{d})\ .$$

**Algorithm III.3**: QRules returns the $Q$-rules of an action $A$ given the reward model $\mathbf{R}$, the current value function $V_t$ and a discount factor $\gamma$. Note that $\tilde{A}$ denotes the action head, where we keep the substitution made by $\text{wp}_i$. We also assume that only legal and completed abstract states $g$ are inserted in *Qrules*.

```
1  initialize Qrules to the empty set.
2  foreach action rule H_i <-p_i:A- B for A do
3  |  foreach v <- V in V_t do
4  |  |  partialQ := {q̃ : Ã <- S | S ∈ wp_i(A, V)}
5  |  |  if S is absorbing then
6  |  |  |  q̃ := R(S)
7  |  |  else
8  |  |  |  q̃ := R(S) + p_i · γ · V_t(V)
9  |  |  if Qrules ≠ ∅ then
10 |  |  |  Qrules := partialQ
11 |  |  else
12 |  |  |  newQ := ∅
13 |  |  |  for all pairs q' : Ã' <- S' ∈ Qrules and q'' : Ã'' <- S'' ∈ partialQ do
14 |  |  |  |  if G := glb(Ã <- S', Ã'' <- S'') exists then
15 |  |  |  |  |  add q : G to newQ with q = q' + q''
16 |  |  |  |
17 |  |  |  Qrules := newQ
18 |  |
19 |
20 return Qrules
```

Thus we add $\mathtt{X} \neq \mathtt{a}, \mathtt{X} \neq \mathtt{c}$ to $S$. The abstract state $S \wedge \mathtt{X} \neq \mathtt{a}, \mathtt{X} \neq \mathtt{c}$ is a legal abstract state. The case that the action caused some effects is covered by the "$\text{mgu}(S'', P)$ *exists*" condition. It is treated analogously.

### 10.4.2 Computing Abstract State Action Values

Given the regressed abstract states and the current abstract state value function $V_t$, we now compute an abstract state-action value function $Q_{t+1}$ according to Algorithm III.3. To do so, we follow a two steps approach:

**(A)** We treat each outcome of an action $A$ as though it would be a single action and compute its abstract state action value.

**(B)** Then, we combine the values of all outcomes to an abstract state action value for $A$, cf. lines 12–17. For the sake of brevity, we will not state constraints in the examples as long as no ambiguities are caused.

For step **(A)**, consider again the first outcome of `move`. The weakest precondition was $\text{wp}_1(\mathtt{move}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}), S') \equiv S_1 \vee S_6$. Because $S_6$, is absorbing, we assign an abstract state action value of 10 for taking action `move`, i.e., $10 : \mathtt{move}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}) \leftarrow S_6$ . The value of $S_1$, however, is dependent on $V_t(S')$, i.e. in our example $V_0$. Assuming a

discount factor of 0.9, this yields $\mathbf{R}(S) + p_1 \cdot 0.9 \cdot V_0(S') = 0 + 0.9 \cdot 0.9 \cdot 10 = 8.1$ , i.e., $8.1 : \texttt{move}(\texttt{a}, \texttt{b}, \texttt{Z}) \leftarrow S_1$ . Doing the same for all other rules in $V_0$ results in:

$\langle a \rangle$ $10 : \texttt{move}(\texttt{X}, \texttt{Y}, \texttt{Z}) \leftarrow \texttt{cl}(\texttt{X}), \texttt{cl}(\texttt{Y}), \texttt{on}(\texttt{a}, \texttt{b}), \texttt{on}(\texttt{X}, \texttt{Z})$
$\langle b \rangle$ $8.1 : \texttt{move}(\texttt{a}, \texttt{b}, \texttt{Z}) \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{b}), \texttt{on}(\texttt{a}, \texttt{Z})$
$\langle c \rangle$ $0.0 : \texttt{move}(\texttt{X}, \texttt{Y}, \texttt{Z}) \leftarrow \texttt{cl}(\texttt{X}), \texttt{cl}(\texttt{Y}), \texttt{on}(\texttt{X}, \texttt{Z})$

For the second outcome of `move`, step **(A)** leads to:

$\langle d \rangle$ $1.0 : \texttt{move}(\texttt{a}, \texttt{X}, \texttt{b}) \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{X}), \texttt{on}(\texttt{a}, \texttt{b})$
$\langle e \rangle$ $1.0 : \texttt{move}(\texttt{X}, \texttt{Y}, \texttt{Z}) \leftarrow \texttt{cl}(\texttt{X}), \texttt{cl}(\texttt{Y}), \texttt{on}(\texttt{a}, \texttt{b}), \texttt{on}(\texttt{X}, \texttt{Z})$
$\langle f \rangle$ $0.0 : \texttt{move}(\texttt{X}, \texttt{Y}, \texttt{Z}) \leftarrow \texttt{cl}(\texttt{X}), \texttt{cl}(\texttt{Y}), \texttt{on}(\texttt{X}, \texttt{Z})$

For step **(B)**, we note that each of these rules describes situations such as *if we are in a state then we can get some value for achieving the i-th outcome of action A*. This information has to be combined to an abstract state action values for $A$. To do so, we select a rule from $\langle a \rangle - \langle c \rangle$, say $\langle b \rangle$, and a rule from $\langle d \rangle - \langle f \rangle$, say $\langle f \rangle$, and check whether we can be in both abstract states at the same time and whether we can apply the same action. In other words, we compute the *greatest lower bound* (glb) of the logical clauses underlying both value rules, see Section 2.2. If the glb exists and it is a legal state, then it is inserted as a new rule, cf. line 15.

In general, computing the glb has high computational costs but the costs can be reduced when assuming that the variables of an action $A$ are the variables occurring in the body and heads of the action's outcomes, i.e., $\text{vars}(A) = (vars(H_i) \cup vars(B))$. In this case, the glb exists only if the actions unify. Thus, we first unify the actions — if possible — and then compute the glb of the resulting bodies, which typically have a much lower number of variables.

The value of the new rule is the sum of values of the combined rules. For $\langle b \rangle$ and $\langle f \rangle$ this yields

$$8.1 : \texttt{move}(\texttt{a}, \texttt{b}, \texttt{X}) \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{b}), \texttt{on}(\texttt{a}, \texttt{X}).$$

In contrast, $\langle b \rangle$ and $\langle d \rangle$ do not result in a new rule.

In our blocks world example, QRules yields the following abstract state action value function when applied to $V_0$, `move` and `absorbing`:

$\langle 1 \rangle$ $10 : \texttt{absorbing} \leftarrow \texttt{on}(\texttt{a}, \texttt{b})$
$\langle 2 \rangle$ $10 : \texttt{move}(\texttt{X}, \texttt{Y}, \texttt{Z}) \leftarrow \texttt{cl}(\texttt{X}), \texttt{cl}(\texttt{Y}), \texttt{on}(\texttt{a}, \texttt{b}), \texttt{on}(\texttt{X}, \texttt{Z})$
$\langle 3 \rangle$ $8.1 : \texttt{move}(\texttt{a}, \texttt{b}, \texttt{X}) \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{b}), \texttt{on}(\texttt{a}, \texttt{X})$
$\langle 4 \rangle$ $0.0 : \texttt{move}(\texttt{X}, \texttt{Y}, \texttt{Z}) \leftarrow \texttt{cl}(\texttt{X}), \texttt{cl}(\texttt{Y}), \texttt{on}(\texttt{X}, \texttt{Z})$

Note that we have sorted the $Q$-rules in descending order only for the sake of readability.

### 10.4.3 Computing Abstract State Values

The set of $Q$-rules enables one to compute the next abstract state value function $V_{t+1}$. In contrast to the traditional case, $Q$-rules, i.e., values of abstract state action pairs,

**Algorithm III.4**: VRules returns the value functions $V_{t+1}$ given the $Q$-rules computed from $V_t$ for all actions.

```
1 initialize V_{t+1} to the empty set of V-rules
2 sort Qrules in decreasing order of Q-values
3 while Qrules not empty do
4 |   remove top element d : A ← B of Qrules
5 |   if no other rule d : A' ← B' in Qrules exists such that B' subsumes B then
6 |   |   add d ← B to V_{t+1}
7 |   |   remove all rules d'' ← B'' from Qrules such that B'' is subsumed by B
8 |
9 return V_{t+1}
```

can overlap such as $Q$-rules $\langle 1\rangle$ and $\langle 2\rangle$. To compute abstract state values we make use of the fact that $V_{t+1}(S) = \max_A Q_{t+1}(S, A)$ due to Equation (9.2).

In general, any value-preserving transformation can be applied. Here, we use a simple separate-and-conquer rule learning approach where the rules to learn and the examples to learn from coincide, see VRules in Algorithm III.4. We search for a $Q$-rule $m$ having a maximal $Q$-value among *Qrules*, cf. lines 2 and 4, separate the covered $Q$-rules, cf. lines 4 and 7, and recursively conquer the remaining $Q$-rules by selecting more rules until no $Q$-rules remain, cf. line 3. The main difference is that we select $m$ and add it to $V_{t+1}$ only if there is no other $Q$-rule left in *Qrules* with the same value whose body subsumes the body of $m$, cf. line 5. In our running example, we start with rule $\langle 1\rangle$. Because it is not subsumed by any other rule having the same value, we add $10 \leftarrow \texttt{on(a,b)}$ to $V_1$ and, because it subsumes $\langle 2\rangle$, we remove $\langle 2\rangle$ from *Qrules*. The remaining highest valued rule is $\langle 3\rangle$, and we iterate. After completing, this yields the new value function $V_1$ (constraints listed again):

$$
\begin{aligned}
10 &\leftarrow \texttt{on(a,b)}, \texttt{a} \neq \texttt{b}.\\
8.1 &\leftarrow \texttt{cl(a)}, \texttt{cl(b)}, \texttt{on(a,X)}, \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{X}, \texttt{b} \neq \texttt{X}.\\
0 &\leftarrow \texttt{cl(X)}, \texttt{cl(Y)}, \texttt{on(X,Z)}, \texttt{X} \neq \texttt{Y}, \texttt{X} \neq \texttt{Z}, \texttt{Y} \neq \texttt{Z}.
\end{aligned}
$$

### 10.4.4 ReBel: Relational Bellman Backup Operator

To summarize, the general scheme of ReBel is:

**1)** Compute the weakest precondition of each action outcome for each abstract state in $V_t$ using WeakestPre.
**2a)** Assign to each resulting abstract state–action outcome pair computed in 1) a $Q$-value (QRules), and
**2b)** combine them using the glb.
**3)** Maximize the $Q$-rules to compute $V_{t+1}$ using VRules.

Note that in **2b)**, if there are $n > 1$ many outcomes of an action, then the $Q$-values of the $n$-th outcome are combined with already combined $Q$-values of the $n-1$ previous outcomes. Thus, there are $n-1$ many combinations per action. This

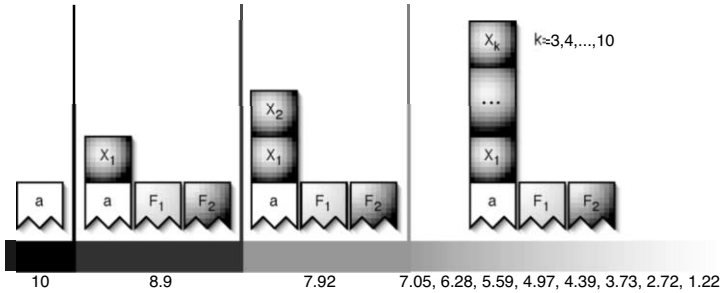



**Figure 10.7. Blocks World Experiment** `cl(a)`**:** Abstract state value function for the `cl(a)` goal after 10 iterations. Inequality constraints are omitted; all variables and constants denote different blocks. $F_i$ can be a block or a floor block. The abstract state value function applies to any number of blocks. Values are rounded to the second digit. States structurally different from the depicted ones get value 0.0 .

may produce many rules. To overcome this, one can adapt the procedure VRules in Algorithm III.4 for maximizing $Q$-rules to *compressing* $Q$-rules: if we are in a state with different currently combined values for compatible actions, then we select only the higher one. This is safe because the higher valued $Q$-rule subsumes the lower valued one. Therefore, it would have been selected in any case later on.

Formally, this Bellman backup requires an infinite number of iterations to converge to $V^*$, cf. Section 10.4.5. In practice, we stop when the abstract value function changes by only a small amount.

### 10.4.5 Experiments

In this section we empirically validate ReBel. We implemented ReBel with compressing *Qrules* in the Prolog system YAP version 4.4.4. and we used the supplemented *constraint handling rules* library [Frühwirth, 1998]. In all experiments we assume a discount factor of 0.9 and a goal reward of 10, i.e., in all other states we receive 0 reward. Only goal states are absorbing. Experiments were run on a 3.1 GHz Linux machine. The running times were estimated using YAP's build-in `statistics(runtime,·)`. We focused on standard examples known from the relational RL literature.

**Blocks World Experiment** `cl(a)`**:** We consider `cl(a)` as goal in our probabilistic blocks world setting. The experiment shows that even on simple problems ReBel is not guaranteed to converge on the structural level.

Figure 10.7 shows the abstract state value function after 10 iterations. It took ReBel roughly 1 minute to iterate ten times. Figure 10.7 highlights that states that are one step further away from the goal get the same value. The value, however, is lower because of the additional block on top of the stack of `a`. Thus, because the number of blocks is not restricted, value iteration will never stop.

**Proposition 10.15** *Abstraction does not guarantee convergence of relational value iteration in infinite domains because an infinite number of abstract states can be required.* ○

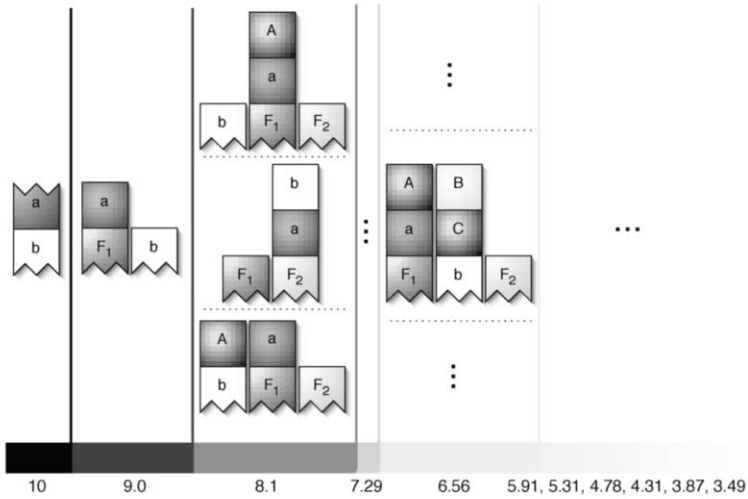



**Figure 10.8. Blocks World Experiment** on(a, b)**:** Parts of the abstract value function for on(a, b) after 10 iterations (values rounded to the second digit). Inequality constraints are omitted; all variables and constants denote different blocks. $F_i$ can be a block or a floor block. The abstract state value function applies for any number of blocks. States more than 10 steps away from the goal get value 0.0.

This is interesting, because infinite state spaces easily arise when relational representations are used and relational abstraction was hoped to be a solution. Nevertheless, relational value iteration can converge even for infinite domains as our third experiment will show.

**Blocks World Experiment** on(a, b)**:** We consider the goal on(a, b) in a deterministic blocks world because it is reported to be a hard problem for model-free relational RL (RRL) approaches [Džeroski et al., 2001, Driessens and Ramon, 2003]. For instance, Driessens and Ramon [2003] report that on average the learned policies did not reach optimal performance even for 5 blocks.

Using the same experimental set-up as in our first experiment but a deterministic move action, ReBel computed $V_{10}$ in less than 12 minutes. The abstract value function is partially shown in Figure 10.8. Because the move action is deterministic, $V_{10}$ is optimal for 10 blocks (more than 58 million ground states). The optimal policy can directly be extracted by computing the maximizing $Q$-rules for each abstract state. In our example, this results in removing the top elements from the stacks on top of a and b. However, to compactly represent this strategy, one needs to define the predicate ontop, which was not present ReBel's experiments but in the experiments Driessens and Ramon [2003] reported on. The policy based on ReBel is optimal no matter how many blocks there are.

**Blocks World Experiment with Multiple Outcomes:** We investigate the move action having three possible outcomes: dropping the block on the intended one with probability 0.8, dropping it on some other block with probability 0.1, or staying

in the current state with probability 0.1. The abstract state value functions for $\texttt{on}(\texttt{a},\texttt{b})$ with a discount factor 0.9 and a goal reward of 10 after three iterations was:

(**1**) $10.00 \leftarrow \texttt{on}(\texttt{a},\texttt{b}), \texttt{a} \neq \texttt{b}$

(**2**) $8.73 \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{b}), \texttt{cl}(\texttt{A}), \texttt{on}(\texttt{a},\texttt{B}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{A} \neq \texttt{B}$

(**3**) $7.41 \leftarrow \texttt{cl}(\texttt{b}), \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{on}(\texttt{a},\texttt{C}), \texttt{on}(\texttt{b},\texttt{a}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{B} \neq \texttt{C}$

(**4**) $7.05 \leftarrow \texttt{cl}(\texttt{b}), \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{on}(\texttt{a},\texttt{C}), \texttt{on}(\texttt{A},\texttt{a}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{B} \neq \texttt{C}$

(**5**) $7.05 \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{on}(\texttt{a},\texttt{C}), \texttt{on}(\texttt{A},\texttt{b}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{B} \neq \texttt{C}$

(**6**) $4.72 \leftarrow \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{cl}(\texttt{C}), \texttt{on}(\texttt{a},\texttt{D}), \texttt{on}(\texttt{b},\texttt{a}), \texttt{on}(\texttt{A},\texttt{b}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{a} \neq \texttt{D}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{b} \neq \texttt{D}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{A} \neq \texttt{D}, \texttt{B} \neq \texttt{C}, \texttt{B} \neq \texttt{D}, \texttt{C} \neq \texttt{D}$

(**7**) $4.12 \leftarrow \texttt{cl}(\texttt{b}), \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{on}(\texttt{a},\texttt{C}), \texttt{on}(\texttt{b},\texttt{D}), \texttt{on}(\texttt{D},\texttt{a}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{a} \neq \texttt{D}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{b} \neq \texttt{D}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{A} \neq \texttt{D}, \texttt{B} \neq \texttt{C}, \texttt{B} \neq \texttt{D}, \texttt{C} \neq \texttt{D}$

(**8**) $4.12 \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{on}(\texttt{a},\texttt{C}), \texttt{on}(\texttt{C},\texttt{b}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{B} \neq \texttt{C}$

(**9**) $3.73 \leftarrow \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{cl}(\texttt{C}), \texttt{on}(\texttt{a},\texttt{D}), \texttt{on}(\texttt{A},\texttt{b}), \texttt{on}(\texttt{C},\texttt{a}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{a} \neq \texttt{D}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{b} \neq \texttt{D}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{A} \neq \texttt{D}, \texttt{B} \neq \texttt{C}, \texttt{B} \neq \texttt{D}, \texttt{C} \neq \texttt{D}$

(**10**) $3.73 \leftarrow \texttt{cl}(\texttt{b}), \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{on}(\texttt{a},\texttt{C}), \texttt{on}(\texttt{A},\texttt{D}), \texttt{on}(\texttt{D},\texttt{a}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{a} \neq \texttt{D}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{b} \neq \texttt{D}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{A} \neq \texttt{D}, \texttt{B} \neq \texttt{C}, \texttt{B} \neq \texttt{D}, \texttt{C} \neq \texttt{D}$

(**11**) $3.73 \leftarrow \texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{on}(\texttt{a},\texttt{C}), \texttt{on}(\texttt{A},\texttt{D}), \texttt{on}(\texttt{D},\texttt{b}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{A}, \texttt{a} \neq \texttt{B}, \texttt{a} \neq \texttt{C}, \texttt{a} \neq \texttt{D}, \texttt{b} \neq \texttt{A}, \texttt{b} \neq \texttt{B}, \texttt{b} \neq \texttt{C}, \texttt{b} \neq \texttt{D}, \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{A} \neq \texttt{D}, \texttt{B} \neq \texttt{C}, \texttt{B} \neq \texttt{D}, \texttt{C} \neq \texttt{D}$

(**12**) $0.00 \leftarrow \texttt{cl}(\texttt{A}), \texttt{cl}(\texttt{B}), \texttt{cl}(\texttt{C}), \texttt{on}(\texttt{A},\texttt{D}), \texttt{A} \neq \texttt{B}, \texttt{A} \neq \texttt{C}, \texttt{A} \neq \texttt{D}, \texttt{B} \neq \texttt{C}, \texttt{B} \neq \texttt{D}, \texttt{C} \neq \texttt{D}$

Observe the exceptional values for states where $\texttt{on}(\texttt{b},\texttt{a})$ is true. This is due to the second outcome of the `move` action. For instance, for the state in line (**3**), both the first and the second outcome take the agent to the same state, namely where $\texttt{cl}(\texttt{a}), \texttt{cl}(\texttt{b})$ holds. In other words, we have a probability of 0.9 of getting to the state in line (**2**). Therefore, we get a slightly higher value than in the states listed in lines (**4**) and (**5**). For the latter ones, the second outcome does not lead to the same states as the first outcome. It might happen that we drop block `A` on top of `b` (by accident). Therefore, we only have a probability of 0.8 to reach the state in line (**2**). This phenomenon occurs later on again, see line (**6**).

**Load-Unload Experiment:** Our final experiment considers the logistics domain, which Boutilier et al. [2001] solved semi-automatically. The domain consists of cities, trucks and boxes. Boxes can be loaded onto and unloaded from trucks, and trucks can be driven between cities. The predicate $\texttt{on}(\texttt{B},\texttt{T})$ denotes that a box `B` is on the truck `T`, $\texttt{bin}(\texttt{B},\texttt{C})$ denotes that a box `B` is in some city `C` and $\texttt{tin}(\texttt{T},\texttt{C})$ denotes that a truck `T` is in city `C`. The actions that can be performed are: $\texttt{load}(\texttt{B},\texttt{T})$ and $\texttt{unload}(\texttt{B},\texttt{T})$ specifying how a box `B` can be loaded onto or loaded from a truck `T` and $\texttt{drive}(\texttt{T},\texttt{C})$ specifying that the truck `T` is driven to city `C`. The actions in this domain have probabilistic effects. The probability of failing a `load` or `unload` action, i.e., staying in the current state, depends on whether it rains or not, denoted by `rain`. The action

| | $V_t$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **abstract states** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| bin(b, p). | **10.000** | 10.000 | 10.000 | 10.000 | 10.000 | 10.000 | 10.000 | 10.000 | 10.000 | 10.000 |
| tin(A, p), on(b, A), not_rain. | **8.100** | **8.829** | **8.895** | **8.901** | 8.901 | 8.901 | 8.901 | 8.901 | 8.901 | 8.901 |
| tin(A, p), on(b, A), rain. | **6.300** | **8.001** | **8.460** | **8.584** | **8.618** | **8.627** | **8.629** | **8.630** | 8.630 | 8.630 |
| tin(A, B), on(b, A), not_rain. | | **7.290** | **7.946** | **8.005** | **8.010** | **8.011** | 8.011 | 8.011 | 8.011 | 8.011 |
| tin(A, B), on(b, A), rain. | | **5.670** | **7.201** | **7.614** | **7.726** | **7.756** | **7.764** | **7.766** | **7.767** | 7.767 |
| tin(A, B), bin(b, B), not_rain. | | | **5.905** | **6.968** | **7.111** | **7.128** | **7.130** | **7.131** | 7.131 | 7.131 |
| tin(A, B), bin(b, B), rain. | | | **3.572** | **5.501** | **6.282** | **6.563** | **6.658** | **6.689** | **6.699** | **6.702** |
| tin(A, B), bin(b, C), not_rain. | | | | **5.314** | **6.271** | **6.400** | **6.416** | **6.417** | **6.418** | 6.418 |
| tin(A, B), bin(b, C), rain. | | | | **3.215** | **4.951** | **5.654** | **5.907** | **5.993** | **6.020** | **6.029** |
| tin(A, B). | **0.000** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

**Table 10.1. Load-Unload Experiment:** The $t$-th column shows the abstract state value function after the $t$-th iteration. When no value is given, the abstract state has value 0.0. Bold numbers highlight changing values from one iteration to the next one.

specification is as follows (we omit the failing specifications for the sake of brevity):

$$\begin{aligned}
\texttt{bin}(\texttt{B},\texttt{C}),\texttt{tin}(\texttt{T},\texttt{C}),R &\xleftarrow{pr:\texttt{unload}(\texttt{B},\texttt{T})} \texttt{on}(\texttt{B},\texttt{T}),\texttt{tin}(\texttt{T},\texttt{C}),R\\
\texttt{on}(\texttt{B},\texttt{T}),\texttt{tin}(\texttt{T},\texttt{C}),R &\xleftarrow{pr:\texttt{load}(\texttt{B},\texttt{T})} \texttt{bin}(\texttt{B},\texttt{C}),\texttt{tin}(\texttt{T},\texttt{C}),R\\
\texttt{tin}(\texttt{T},\texttt{C}'),\texttt{C}\neq\texttt{C}' &\xleftarrow{1.0:\texttt{drive}(\texttt{T},\texttt{C}')} \texttt{tin}(\texttt{T},\texttt{C})
\end{aligned}$$

where the probability $pr$ is 0.9 if $R$ is `rain` and 0.7 if $R$ is `not_rain`. To correctly handle the *explicit negation* we used for `rain`, we provided $\texttt{false} \leftarrow \texttt{rain},\texttt{not\_rain}$ as constraint. The goal in this domain is to get some box `b` in `p` where `p` stands for *Paris*, i.e., in $\texttt{bin}(\texttt{b},\texttt{p})$ we get a reward of 10.

ReBel ran for less than 6 seconds to compute the results summarized in Table 10.1. In contrast to the blocks world examples, the solution converges both at the *value* level and at the *structural* level. For instance, consider the situation in which a truck is in a city different from Paris and the box is there too. Then, it will take three steps ($\texttt{load}-\texttt{drive}-\texttt{unload}$) to reach the goal state and the state value in $V_{10}$ is 6.702 in case it rains. The abstract state value function applies no matter how many trucks, boxes and cities are present.

# Future Work

Our work on relational variants of Markov decision processes and on relational reinforcement learning leaves a lot of space for improvements. The languages of Markov decision programs and of abstract value and policy functions are somewhat restrictive. Negation and $\forall$-quantification within abstract states and value functions and using background knowledge not only for checking for illegal states but also for describing abstract states and value functions would significantly broaden the application perspective. Within generalized relational policy iteration, relational state splitting rules specific for Markov decision processes could be employed. Other model-free and model-based relational reinforcement learning techniques could be devised and explored. Our convergence result for $RTD(\lambda)$ considers the evaluation problem only and could be extended to the complete generalized relational policy iteration cycle. A first step into that direction has been provided by Ramon [2005], who proved convergence results for RRL-TG. In general, convergence results for other relational techniques such as relational instance based regression [Driessens and Ramon, 2003] could be provided. Some of them, however, are quite challenging. For instance, proving convergence of logical Q-learning [Kersting and De Raedt, 2003] requires to investigate action aggregation. A major drawback of ReBel that we have discussed is that ReBel involves in each iteration operations over the entire set of abstract states known at that time. If the abstract state set is very large, then even a single sweep can be prohibitively expensive. Asynchronous dynamic programming approaches [Sutton and Barto, 1998] back up the values of states in any order whatsoever, using whatever values of other states happen to be available. In turn, they can avoid hopelessly long computations before

improving a policy. Fischer [2005] provided a first asynchronous relational value iteration approach. Fischer's results are promising but also show a higher computational burden. Advanced relational asynchronous approaches would also lead to a better understanding of the connection between model-based and model-free relational reinforcement learning approaches.

The overall goal of relational reinforcement learning is to learn abstract policies. Most current relational reinforcement learning techniques, including the one presented in this thesis, are value-based. Value functions, however, can be very complex for large problems. In the relational case, value functions can even be infinite, i.e., the values of infinitely many abstract states have to be represented. Policies, in contrast, have typically a simpler form. Relational, i.e. abstract policies generalize over objects and values and — by employing background knowledge — can even be described with a finite set of abstract states when there are infinitely abstract states with different values. Therefore, one promising future direction is to search directly in the space of abstract policies. In [Cocora et al., 2005], we proposed to generalize ground policies using logical decision trees [Blockeel and De Raedt, 1998]. The experimental results show that this approach performs better than generalizing the corresponding value functions using relational regression trees [Mausam and Weld, 2003]. For traditional reinforcement learning, direct search in policy space has been proven successful using policy gradient methods [Williams, 1992, Sutton et al., 2000]. Thus, much work on relational reinforcement learning remains to be done.

# Conclusions

We have introduced the framework of Markov decision programs. Markov decision programs integrate MDPs with constraint logic programs. They allows one to compactly and declaratively represent complex (relationally factored) MDPs. Furthermore, it allows one to gain insights into relational reinforcement learning approaches. More precisely, we introduced abstract policies for Markov decision programs and *generalized relational policy iteration* (GRPI), which is a general scheme for learning abstract policies. We then devised two relational reinforcement learning methods that update the values of relationally grouped states simultaneously. They essentially adapt the probabilistic learning from interpretations to the reinforcement learning setting.

Both algorithms have lead to novel insights into relational MDPs and relational reinforcement learning. First, it has been shown that value-based methods for relational MDPs may not converge because an infinite number of abstract states has to be represented. This implies that RRL-TG may require an infinite logical regression tree in order to converge to exact values. Here, approximative approaches such as the presented RTD($\lambda$) are useful, as they allow one to cut the regression tree at any level and to estimate the best values one can achieve at that abstraction level. The convergence of RTD($\lambda$) has been proven. Thus, approximation is not only an interesting feature, but in some cases also a necessity for successful relational

reinforcement learning. Second, we highlighted that the use of background knowledge may also enable the learning of optimal policies. Depending on the representation of the problem, one can or cannot learn the optimal policy. Therefore, using background knowledge in relational MDPs is also not only an interesting feature, but in some cases also a necessity for successful learning. In this way, we have given an explanation for and confirmed some of the experimental insights of the early relational reinforcement learning work.

# Related Work

Relational reinforcement learning is an emerging research field. Kurt Driessens's Ph.D. thesis [2004] and the working notes of the two ICML workshops on relational reinforcement learning [Tadepalli et al., 2004, Driessens et al., 2005] give a good overview of the field.

One of the earliest related works is probably that by Baum [1999], who reported on solving blocks worlds with up to 10 blocks using related techniques. However, the language is domain-dependent and is not based on logic programming. Kaelbling et al. [2001] and Finney et al. [2002] investigated propositionalization methods in relational domains. They experimentally studied the intermediate language of *deictic representations* (DRs). DRs avoid enumerating the domain by using variables such as *the-block-on-the-floor*. Although DRs have led to impressive results McCallum [1995], Whitehead and Ballard [1991], Finney et al.'s [2002] results show that DR may degrade learning performance within relational domains. According to Finney et al., *relational reinforcement learning* such as RRL-RT Džeroski et al. [2001] is one way to effectively learning in domains with objects and relations. The $Q$-function in RRL-RT is approximated using a relational regression tree learner. Although the experimental results are interesting, Džeroski et al. did not give a theoretical argument why RRL-RT works. We provide some new insights on this. Several other relational learners for function approximation have been studied within reinforcement learning [Lecoeuche, 2001, Driessens and Ramon, 2003, Gärtner et al., 2003, Driessens, 2004]. Again, these works presented impressive empirical but no theoretical results. Other ones applied $Q$-learning based on pre-specified abstract state spaces. Kersting and De Raedt [2003] investigated pure $Q$-learning, Van Otterlo [2004] learned the $Q$-function via learning the underlying transition model. The work on $RTD(\lambda)$ complements Kersting and De Raedt's [2003] and Van Otterlo's [2004] approaches as both did not provide convergence proofs. Croonenborghs et al. [2004] learned partial relational models of the world using Bayesian logic programs. These partial models are then used to speed up a relational reinforcement learner. Similar, Zettlemoyer et al. [2005] learned probabilistic STRIPS-like action definitions from observations only and evaluated them within MDPs. Yoon et al. [2002] introduced a model-based method for upgrading abstract policies from small RMDPs to larger ones. Fern et al. [2004] extended this

work on upgrading learned policies for small relational MDPS (RMDPs) with *approximated policy iteration*. Finally, Guestrin et al. [2003] specified relationally factored MDPs based on probabilistic relational models [Friedman et al., 1999] but not in a reinforcement learning setting. In contrast to Markov decision programs, relations do not change over time. This assumption does not hold in many domains such as the blocks world. Finally, the work by Dietterich and Flann [1997] is also concerned with generalizing Bellman backups but no relational representation is used.

For model-based approaches to relational reinforcement learning, there has been a surprising lack of research on *exact* solution methods. From a more general point of view, our approach is closely related to *decision theoretic regression* (DTR) [Boutilier et al., 1999]. In DTR, state spaces are characterized by a number of random variables and the domain is specified using logical representations of actions that capture the regularities in the effects of actions. Because 'existing DTR algorithms are all designed to work with *propositional* representations of MDPs', Boutilier et al. [2001] proposed *first order DTR* (FODTR), which is a probabilistic extension of Reiter's *situation calculus*. ReBeL relates to this in that it is also a model-based exact solution method for RMDPs. One key difference with ReBel is that situation calculus is very expressive and as a consequence it is harder to simplify the logical descriptions of the abstract value functions and abstract states that are obtained. This may also explain why — to the best of the author's knowledge — that approach has not been fully implemented and experimented with until recently [Sanner and Boutilier, 2004, 2005]. In contrast, because of the use of a simpler logical language, the simplification in ReBel is computationally feasible. As shown in the experiments, ReBel successfully and fully automatically implements relational value iteration. Recently, Sanner and Boutilier [2005] proposed to represent and compute value function of FODTRs as a linear combination of first-order basis functions and using a first-order generalization of approximate linear programming techniques for propositional MDPs. Furthermore, Boutilier et al. [2001] and Sanner and Boutilier [2005] assumed that the model is given whereas we have also investigated model-free learning methods.

Großmann et al. [2002] combined MDPs with the fluent calculus. Based on this representation language Karabaev and Skvortsova [2005] recently presented a fully automated heuristic search algorithm called $FOLAO^*$ for solving models described within Grossmann *et al.*'s framework. $FOLAO^*$ is model-based and consists of two phases that alternate until a complete solution is found. First, it expands the best partial policy and evaluates the states on its fringe using an admissible heuristic function. Then it performs dynamic programming on the states visited by the best partial policy, to update their values and possibly revise the current best partial policy.

Markov decision programs are also related to Poole's [1997] independent choice logic, see also the related work section of Part I. Poole, however, does not consider the learning problem.

The idea of solving large MDP by a reduction to an equivalent, smaller MDP is also discussed e.g. in [Dearden and Boutilier, 1997, Givan et al., 2003]. However, these works do not investigate relational nor first order representations have. Kim and Dean [2003] investigated model-based RL based on non-homogenous partitions of propositional, factored MDPs. Furthermore, there has been great interest in abstraction on other levels than state spaces. Abstraction over time [Sutton et al., 1999] or primitive

actions [Dietterich, 2000, Andre and Russell, 2001] are useful ways to abstract from specific sub-actions and time. This research is orthogonal to the one pursued in this part and could be applied to Markov decision programs in the future.

# Finale

The thesis has demonstrated that by exploiting inductive logic programming, we can develop a general framework for and several approach to statistical relational learning, which highlight the benefits of relational and logical abstraction through variables and unification. We will now conclude the thesis. It provides a summary of the frameworks presented in this thesis, along with a discussion of general directions of future research.

This page intentionally left blank

§ 11

# Summary

Statistical relational learning addresses one of the central open questions of artificial intelligence: the combination of relational and first-order logic with principled probabilistic and statistical approaches to inference and learning. In this thesis, we first introduced probabilistic ILP as a general framework for statistical relational learning, and then, we presented several probabilistic ILP settings and approaches.

Probabilistic ILP views statistical relational learning from an inductive logic programming perspective. It makes abstraction of specific probabilistic relational and first order logical representations and inference and learning algorithms. Therefore, it can be used for obtaining an appreciation of the differences and similarities among various statistical relational learning frameworks and formalisms that have been contributed to date. In particular, the distinction between learning from entailment, learning from interpretations, and learning from proofs can be used for clarifying the relation among the relational and logical upgrades of Bayesian networks (such as probabilistic-logic programs, probabilistic relational models, relational Bayesian networks, Bayesian logic programs, relational Markov models, and Markov logic networks) and probabilistic grammars (such as PRISM, stochastic logic programs, relational Markov models, and logical hidden Markov models). Furthermore, principles of both statistical learning and inductive logic programming (or multi-relational data mining) are employed for learning the parameters and structure.

The first probabilistic ILP approach we introduced, Bayesian logic programs, considered probabilistic learning from interpretations. Bayesian logic programs treat ground atoms as random variables and view the immediate consequence operator as the probabilistic dependency relation. Scooby, the learning algorithm we presented, combines methods for scoring Bayesian networks, such as EM and gradient-based methods, with techniques employed in the inductive logic programming system Claudien. The approach highlights the benefits of an inductive logic programming view on statistical relational learning: instead of manipulating single edges, refinement operators work on bunches of edges and, hence, make larger steps in the search space. Language bias (such as types, modes, and considering constant free programs only) further constrain the search space. Finally, declarative background knowledge can always easily be represented and used.

Then, we considered probabilistic ILP over time. Even though time can be considered as yet another Bayesian predicate in Bayesian logic programs, such a view would not solve the inference problem: the set of possible state trajectories grows exponentially over time. Therefore, we introduced logical hidden Markov models, which deal with sequences of logical atoms. The experimental results have shown the benefits of logical abstraction through variables and unification: high compression in the number of parameters with good estimation and predictive performance. The learning method for selecting logical hidden Markov models from data combines generalized EM, which optimizes parameters, with ILP refinement operators for structure search.

The third class of approaches that we explored addressed discriminative classifi-

cation. We showed how to apply the generic mechanism of Fisher kernels, that is, we showed how to exploit Bayesian logic programs and logical hidden Markov models within discriminative classifiers such as support vector machines. For discrimination, the resulting relational Fisher kernels have been experimentally shown to improve upon the generative model alone with only little additional computational costs. In principle, any attribute-value based classifier can be used but kernel methods such as support vector machines are most naturally suited. While we have used classification to guide the development of relational Fisher kernels, the results — as for propositional Fisher kernels — are directly applicable to other tasks such as regression and clustering, all of which can easily exploit metric relations among the examples defined by the Fisher kernel. This is a first step into the direction of discriminative, probabilistic ILP.

The final probabilistic ILP approaches we introduced were concerned with Markov decision programs, which employ constraint logic programming to compactly and declaratively represent relationally structured Markov decision processes and, hence, extend probabilistic ILP over time towards decision-theoretic reasoning and planning. We used Markov decision programs to make first steps towards a theory of relational reinforcement learning. We devised relational upgrades of temporal difference learning and value iteration and proved their convergence. Furthermore, we placed them into the general context of generalized relational policy iteration for solving relationally structured Markov decision processes. ILP concepts such as $\theta$-subsumption and the greatest lower bound were used to maintain relational descriptions of the world.

For all approaches, we presented experimental results that show their practical relevance and computational feasibility and that exploiting relational and logical structure for learning probabilistic models is beneficial.

§ 12

# Conclusions

Relational and first order logical reasoning, probabilistic and statistical reasoning, and machine learning are research fields on their own rights. Nowadays, they are becoming increasingly intertwined. A major driving force is the explosive growth in the amount of heterogeneous data that is being collected in the business and scientific world. Example domains include bioinformatics, transportation systems, communication networks, social network analysis, citation analysis, robotics, among others. They provide uncertain information about relational worlds. Techniques for descriptive and predictive machine learning tasks, for modeling dynamic environments, and for making complex decisions within these worlds are needed and have the potential to lay the foundation for the next generation of artificial intelligence.

Indeed, statistical relational learning is quite central to these kind of domains. This thesis has described our attempt to approach statistical relational learning from an inductive logic programming point of view. The main idea is to **group together similar states** respectively **similar random variables** together using logical atoms and queries. States of random variables are identified with ground

atoms such as emacs(lohmms, tex) or with sets of ground atoms, i.e., interpretations such as cl(a), on(a, b), block(a), block(b). Likewise, ground atoms can also be treated as random variables with associated sets of possible states. For instance, carrier(ann) represents whether Ann is a carrier of a disease or not. Now, logical variables allow one to **make abstraction of specific symbols**. Using the logical concepts of atoms and queries, states and random variables are grouped together because they have the same conditional probability distribution, similar state values, the same (optimal) action, or the same transition model. For instance, carrier(Person) denotes the disease carrier probability for all persons and cl(Block), on(Block, b) describes all blocks world situations, where there is a clear block on top of block b. Unification allows one to **share the knowledge among states and random variables** using clauses or STRIPS operators. Consider carrier(Person)|mother(Mother, Person), carrier(Mother), which expresses the influence of the mother's chance of being a carrier on her children. probabilities, or on(X, Y), cl(X), cl(Z) $\xleftarrow{0.9:\texttt{move(X,Y,Z)}}$ cl(X), cl(Y), on(X, Z), which specifies moving blocks in the blocks world. At running time, the **abstract knowledge must be recombined** using some logical inference mechanism such as backward chaining or resolution. This logical reasoning mechanism bears additional computational costs but it also makes statistical relational models more flexible, context-aware, and offers the full power of logical reasoning. Indeed, there are typically multiple clauses or STRIPS operators such as carrier(Person)|father(Father, Person), carrier(Father), which yield **multiple**, even **conflicting informations** about single states or random variables. To **integrate** or **resolve** them, combining rules, aggregate functions, preference orders, or greatest lower bounds can be used; they basically **partition** the states space (respectively the space of random variables). These partitions, in turn, are the key to adapt statistical learning methods such as the EM and the Viterbi algorithm, temporal difference learning, and value iteration: **treat each group member as an independent experiment**. Finally, **relational and logical abstraction induce a structure on the hypothesis space**, which can be employed using traditional ILP principles: $\theta$-subsumption implies a generality order among hypotheses, which can be traversed by applying refinement operators traditionally employed within ILP. Thus, probabilistic ILP approaches typically combine statistical learning approaches for optimizing parameters and scoring hypotheses with structure search for model selection using ILP principles. As it turned out in the experiments, relational and logical abstraction indeed yield compact models and makes learning more robust.

§ 13

# Future Work

Statistical relational learning is a new research field and shows many opportunities for future research. The introduced settings for probabilistic ILP are only a first step towards a theory of statistical relational learning. Further research on the semantics of probabilistic relational and first order representations, the expressive power of

alternative formalisms and primitives, scoring functions unifying statistical learning and inductive logic programming scores, optimal refinement operators, the trade-off between expressive power and computational cost, efficient data structures, and the convergence and the complexity of learning could be conducted. Let us now address some of these issues in more detail.

A **theory of statistical relational learning / probabilistic ILP** could be work out in the future. A diversity of statistical relational learning approaches haven been developed within the last ten years. No clear and generally accepted understanding of the relative advantages and limitations of different techniques has yet emerged. One way to gain such an understanding is to provide mappings between different statistical relational learning approaches. Mappings naturally facilitate the transfer of techniques developed within different frameworks. Although some mappings have been already proposed in the literature — mainly when a new framework has been introduced, see e.g. the related work section for Bayesian logic programs and [Kersting and De Raedt, 2001b, Santos Costa et al., 2003a, Puech and Muggleton, 2003, Vennekens and Verbaeten, 2003, Domingos and Richardson, 2004, Vennekens et al., 2004, Jäger, 2005, Fierens et al., 2005] — no general framework has been developed yet. In the future, a hierarchy among the statistical relational learning approaches could be developed that takes the complexities of the mappings and the type of queries a framework supports into accounts. Such complexity-hierarchies lay at the heart of theoretical computer science. In computational complexity, for instance, the polynomial-time hierarchy has had a major impact in many area. Equivalence notations within a particular SRL approach but also among different ones would be interesting.

In general, discovering the common methods underlying efficient statistical relational learning algorithms and identifying their computational impediments, is a major future research issue. The speed up of statistical learning through relational abstraction has been reported only empirically. Sample complexities results such as relational PAC learning models could be established. Sample complexity bounds could also be employed to prune the search space. Furthermore, in computational learning theory, an important issue is the so-called bias-variance trade-off, which shows that a model with a high degree of freedom may have poor generalization capabilities. Combining statistical models with relational representations even increases the degrees of freedom. It seems to be that there is a *probability - structure* trade-off. Right now, no clear understanding of how much probability and how much relational and logical expressiveness is needed. The research on inductive logic programming has already shown that other biases such as language and search bias and background knowledge are a key to good generalization performance. Recently, Landwehr et al. [2005] and Davis et al. [2005] have started to explore the *probability - structure* trade-off. They combined ILP system with simple types of Bayesian networks such as Naïve Bayes and tree-augmented networks. The results are promising. Landwehr et al.'s NFOIL system outperforms well-developed ILP systems on ILP benchmark data sets. Davis et al.'s SAYU system outperformed Domingos and Richardson's Markov logic networks on a web-page classification task.

So far, general **efficient data pre-processing and data structures** have not been investigated extensively. Probabilistic ILP approaches usually evaluate a large number of probabilistic queries on the data set. The subsumption lattice among these

queries typically shows that these queries are very similar. As a consequence, independent execution of all queries may involve a lot of redundant computation. Future work could focus on reducing the running time by exploiting redundancy among queries and data cases. One promising approach is to employ (inductive) logic programming techniques. For instance, [Sato and Kameya, 2001, Kameya et al., 2004] have shown that tabling of queries can greatly speed up inference within PRISM programs. The idea of tabling is as follows. During the execution of a query, each subgoal $S$ is registered in a table the first time it is called, and unique answers to $S$ are added to the table as they are derived. So far, tabling has only be applied in the context of parameter estimation of PRISM [Sato and Kameya, 2001]. Using tabling while structure learning is an interesting direction for future work. One could also investigate query packs [Blockeel et al., 2002], which structure sets of similar queries, and distribution preserving transformations of statistical relational learning models. Such transformations have been shown to speed up the execution of queries for logical programs considerably [Santos Costa et al., 2003b]. Alternatively, one might want to develop relational variants of AD-trees [Moore and Lee, 1998]. AD-trees are an efficient data structure for caching sufficient statistics, i.e., ground counts and have successfully been used within attribute-value based machine learning. Thon [2004] adapted tries to efficiently store pairs of ground atoms and their corresponding counts for estimating the parameters of logical Markov models. The data structure considerably reduced the running time. A similar technique might be used to speed-up sagEM for learning logical hidden Markov models. Recently, Sanner and McAllester [2005] proposed an extension to ADDs, called affine ADDs, capable of compactly representing context-specific, additive, and multiplicative structure within ADDs and applied it to Bayesian network and MDP inference. Lifting efficient data structures to the relational case, where some of the logical or probabilistic inference has been pre-compiled is needed to speed up inference and, in turn, learning.

Most inference algorithms within statistical relational learning, build ground models such as a Bayesian network or a ground trellis for inference and learning. To overcome this, **advanced inference** techniques could be developed. For instance, research towards a lifting theorem for relational and logical probabilistic inference would be useful [Poole, 2003, de Salvo Braz et al., 2005] proposed first approaches into this direction. The work on statistical relational learning has shown that structured representations of probabilistic models can lead to better performance. In general, such representations might also directly lead to novel and better approximate inference algorithm. For instance, relational abstraction might be employed within particle filters. The basic idea of particle filters is Monte Carlo simulation, in which the posterior density is approximated by a set of particles with associated weights. Relational abstraction may help to reduce the number of particles. Particles are shared by states subsumed by the same abstract state. Even (sets of) particles might be represented relationally, for example using decision trees. There is much work remaining to be done on inference and learning for complex dynamic models integrating temporal and relational structure.

Many successful applications of statistical learning such as speech recognition or computer vision call for continuous random variables. Most probabilistic ILP approach today, however, consider random variables with categorical values and multinomial

distributions only [43]. The most common distribution for continuous variables is the Gaussian. Exploring the world of models mixing continuous and discrete valued random variables is expected to yield — as for Bayesian networks — a rich toolbox for modeling complex relational models that upgrade hierarchical mixture of experts, factor analysis, principle component analysis, and independent component analysis, among others [Rosweis and Ghahramani, 1999]. In general, **a shift from general purpose frameworks towards particular tasks** such as discriminative learning, clustering, and semi-supervised learning is an obvious and interesting although challenging future line of research. In this context, combining probabilistic ILP within kernel methods is an attractive research direction.

**Real-Time AI** systems '*are required to work continuously over extended periods of time, interface to the external environment via sensors and actuators, deal with uncertain or missing data, focus resources on the most critical events, handle both synchronous and asynchronous events in a predictable fashion with guaranteed response times, and degrade gracefully*' [Musliner et al., 1995]. Statistical relational learning already addresses many of the real-time AI issues. So far, however, any-time algorithms for inference, learning, and decision making, which provide answers at any point in their execution and the quality of the answer improves with an increase in execution time, have not been developed. Furthermore, real-time heuristic search methods, which interleave planning and plan execution and often decrease the sum of planning and plan-execution time because gathering information early reduces the subsequent amount of planning needed, have not been investigated. Such techniques are likely to have great impact for instance on intelligent web technologies and electronic games.

Although there is a steady growing body of applications, we believe there is still a good deal of work to be done on **applications**. Interesting application areas are web search and mining, bioinformatics, natural language processing, among other. Smart web technologies, for instance relational variants of Rennie and McCallum's [1999] reinforcement learning-based webspiders and of Shani et al.'s [2005] Markov decision process for recommendation systems, are interesting. *Relational robotics*, i.e., the application of (statistical) relational learning within robotics is another promising line of future research. Anguelov et al. [2005] already used relational Markov networks for segmentation of 3D scan data. Recently, In [Triebel et al., 2006], we made Anguelov et al.'s method more robust by compressing scans using kd-trees. Relational Markov networks have also been used to compactly represent object maps [Limketkai et al., 2005] and to estimate trajectories of people [Liao et al., 2005]. We have proposed a generalized relational policy iteration approach to learn relational navigation policies for robots [Cocora et al., 2005]. Relational robotics raises also the questions of using statistical relational learning and reasoning as well as reinforcement learning in the multi-agent context. Here, many interesting problems can be explored, such as communication and cooperation. Agents in these contexts have often been modeled in terms of first-order languages.

Developing further real-world applications of statistical relational learning ap-

[43] Probabilistic relational models and Bayesian logic programs allow to use continuous random variables.

proaches and discovering their limitations is perhaps the best way to find out which of their features and functionalities are really important.

This page intentionally left blank

# Appendix

This page intentionally left blank

# Models for Unix Command and mRNA Sequences *

*... in which the logical hidden Markov models will be presented used to show the benefits of variable sharing and unification and used in the tree-structured mRNA experiments ...*

## A.1 Logical HMM for Unix Command Sequences

The logical HMMs described below model UNIX command sequences triggered by `mkdir`. To this aim, we transformed the original Greenberg data into a sequence of logical atoms over $\texttt{com}, \texttt{mkdir}(\texttt{Dir}, \texttt{LastCom}), \texttt{ls}(\texttt{Dir}, \texttt{LastCom}), \texttt{cd}(\texttt{Dir}, \texttt{Dir}, \texttt{LastCom})$, $\texttt{cp}(\texttt{Dir}, \texttt{Dir}, \texttt{LastCom})$ and $\texttt{mv}(\texttt{Dir}, \texttt{Dir}, \texttt{LastCom})$. The domain of `LastCom` was $\{\texttt{start}, \texttt{com}, \texttt{mkdir}, \texttt{ls}, \texttt{cd}, \texttt{cp}, \texttt{mv}\}$. The domain of `Dir` consisted of all argument entries for $\texttt{mkdir}, \texttt{ls}, \texttt{cd}, \texttt{cp}, \texttt{mv}$ in the original dataset. Switches, pipes, etc. were neglected, and paths were made absolute. This yields 212 constants in the domain of `Dir`. All original commands, which were not $\texttt{mkdir}, \texttt{ls}, \texttt{cd}, \texttt{cp}$, or `mv`, were represented as `com`. If `mkdir` did not appear within 10 time steps before a command $C \in \{\texttt{ls}, \texttt{cd}, \texttt{cp}, \texttt{mv}\}$, $C$ was represented as `com`. Overall, this yields more than 451000 ground states that have to be covered by a Markov model.

The 'unification' logical HMM $U$ basically implements a second order Markov model, i.e., the probability of making a transition depends upon the current state and the previous state. It has 542 parameters and the following structure:

$$\begin{array}{rr} \texttt{com} \leftarrow \texttt{start}. & \texttt{com} \leftarrow \texttt{com}. \\ \texttt{mkdir}(\texttt{Dir}, \texttt{start}) \leftarrow \texttt{start}. & \texttt{mkdir}(\texttt{Dir}, \texttt{com}) \leftarrow \texttt{com}. \\ & \texttt{end} \leftarrow \texttt{com}. \end{array}$$

Furthermore, for each $C \in \{\texttt{start}, \texttt{com}\}$ there are

$$\begin{array}{rr} \texttt{mkdir}(\texttt{Dir}, \texttt{com}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). & \texttt{cd}(\_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). \\ \texttt{mkdir}(\_, \texttt{com}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). & \texttt{cp}(\_, \texttt{Dir}, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). \\ \texttt{com} \leftarrow \texttt{mkdir}(\texttt{Dir}, C). & \texttt{cp}(\texttt{Dir}, \_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). \\ \texttt{end} \leftarrow \texttt{mkdir}(\texttt{Dir}, C). & \texttt{cp}(\_, \_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). \\ \texttt{ls}(\texttt{Dir}, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). & \texttt{mv}(\_, \texttt{Dir}, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). \\ \texttt{ls}(\_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). & \texttt{mv}(\texttt{Dir}, \_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). \\ \texttt{cd}(\texttt{Dir}, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). & \texttt{mv}(\_, \_, \texttt{mkdir}) \leftarrow \texttt{mkdir}(\texttt{Dir}, C). \end{array}$$

* The material here is taken from [Kersting et al., 2006].

together with for each $C \in \{\texttt{mkdir}, \texttt{ls}, \texttt{cd}, \texttt{cp}, \texttt{mv}\}$ and for each $C_1 \in \{\texttt{cd}, \texttt{ls}\}$ (resp. $C_2 \in \{\texttt{cp}, \texttt{mv}\}$)

$$
\begin{array}{rlrl}
\texttt{mkdir}(\texttt{Dir}, \texttt{com}) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{mkdir}(\_, \texttt{com}) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{mkdir}(\_, \texttt{com}) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{com} & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{com} & \leftarrow C_1(\texttt{Dir}, C). & \texttt{end} & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{end} & \leftarrow C_1(\texttt{Dir}, C). & \texttt{ls}(\texttt{From}, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{ls}(\texttt{Dir}, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{ls}(\texttt{To}, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{ls}(\_, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{ls}(\_, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{cd}(\texttt{Dir}, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{cd}(\texttt{From}, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{cd}(\_, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{cd}(\texttt{To}, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{cp}(\_, \texttt{Dir}, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{cd}(\_, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{cp}(\texttt{Dir}, \_, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{cp}(\texttt{From}, \_, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{cp}(\_, \_, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{cp}(\_, \texttt{To}, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{mv}(\_, \texttt{Dir}, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{cp}(\_, \_, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{mv}(\texttt{Dir}, \_, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{mv}(\texttt{From}, \_, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
\texttt{mv}(\_, \_, C_1) & \leftarrow C_1(\texttt{Dir}, C). & \texttt{mv}(\_, \texttt{To}, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C). \\
 & & \texttt{mv}(\_, \_, C_2) & \leftarrow C_2(\texttt{From}, \texttt{To}, C).
\end{array}
$$

Because all states are fully observable, we omitted the output symbols associated with clauses, and, for the sake of simplicity, we omitted associated probability values.

The 'no unification' logical HMM $N$ is the variant of $U$ where no variables were shared such as

$$
\begin{array}{rlrl}
\texttt{mkdir}(\_, \texttt{com}) & \leftarrow \texttt{cp}(\texttt{From}, \texttt{To}, C). & \texttt{ls}(\_, \texttt{cp}) & \leftarrow \texttt{cp}(\texttt{From}, \texttt{To}, C). \\
\texttt{com} & \leftarrow \texttt{cp}(\texttt{From}, \texttt{To}, C). & \texttt{cd}(\_, \texttt{cp}) & \leftarrow \texttt{cp}(\texttt{From}, \texttt{To}, C). \\
\texttt{end} & \leftarrow \texttt{cp}(\texttt{From}, \texttt{To}, C). & \texttt{cp}(\_, \_, \texttt{cp}) & \leftarrow \texttt{cp}(\texttt{From}, \texttt{To}, C). \\
 & & \texttt{mv}(\_, \_, \texttt{cp}) & \leftarrow \texttt{cp}(\texttt{From}, \texttt{To}, C).
\end{array}
$$

Because only transitions are affected, $N$ has 164 parameters less than $U$, i.e., 378.

## A.2 Tree-based logical HMM for mRNA Sequences

The logical HMM processes the nodes of mRNA trees in in-order. The structure of the logical HMM is shown at the end of the section. There are copies of the shaded parts. Terms are abbreviated using their starting alphanumerical; `tr` stands for `tree`, `he` for `helical`, `si` for `single`, `nuc` for `nucleotide`, and `nuc_p` for `nucleotide_pair`.

The domain of *#Children* covers the maximal branching factor found in the data, i.e., $\{[\texttt{c}], [\texttt{c}, \texttt{c}], \ldots, [\texttt{c}, \texttt{c}, \texttt{c}, \texttt{c}, \texttt{c}, \texttt{c}, \texttt{c}, \texttt{c}, \texttt{c}]\}$; the domain of *Type* consists of all types occurring in the data, i.e., $\{\texttt{stem}, \texttt{single}, \texttt{bulge3}, \texttt{bulge5}, \texttt{hairpin}\}$; and for *Size*, the domain covers the maximal length of a secondary structure element in the data, i.e., the longest sequence of consecutive bases respectively base pairs constituting a secondary structure element. The length was encoded as $\{\texttt{n}^1(\texttt{0}), \texttt{n}^2(\texttt{0}), \ldots, \texttt{n}^{13}(\texttt{0})\}$ where $\texttt{n}^m(\texttt{0})$ denotes the recursive application of the functor `n` $m$ times. For *Base* and *BasePair*, the domains were the 4 bases respectively the 16 base pairs. In total, there are 491 parameters.

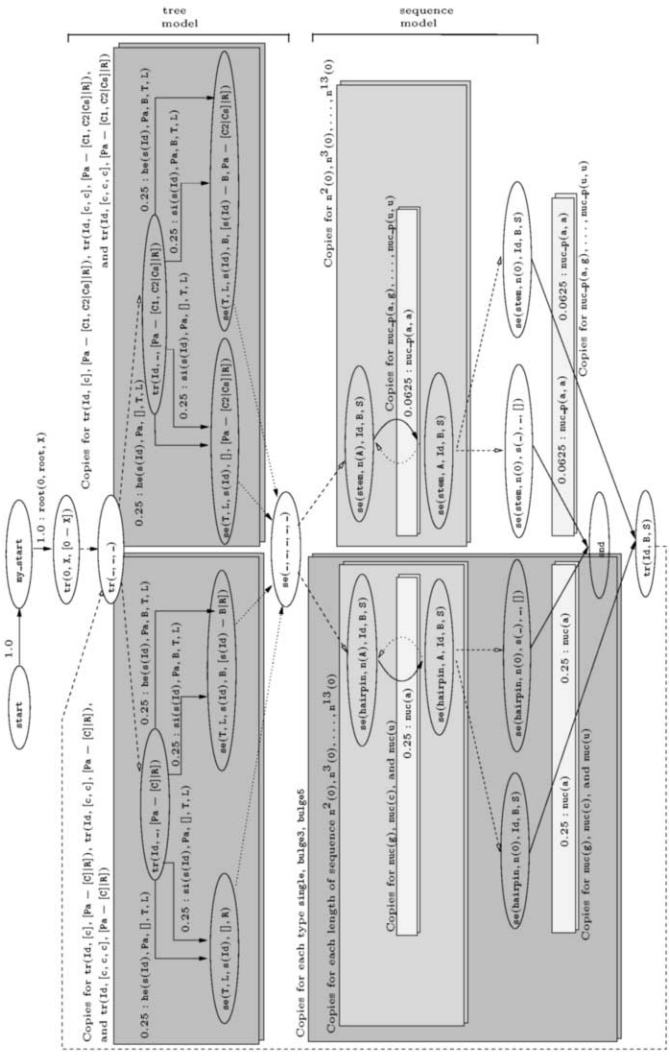



**Figure A.1.** The mRNA logical HMM structure. The symbol _ denotes anonymous variables which are read and treated as distinct, new variables each time they are encountered. There are copies of the shaded part. Terms are abbreviated using their starting alphanumerical; `tr` stands for `tree`, `se` for `structure_element`, `he` for `helical`, `si` for `single`, `nuc` for `nucleotide`, and `nuc_p` for `nucleotide_pair`.

This page intentionally left blank

# Bibliography


S. P. Abney. Stochastic Attribute-Value Grammars. *Computational Linguistics*, 23 (4):597–618, 1997.

S. P. Abney, D. A. McAllester, and F. Pereira. Relating probabilistic grammars and automata. In R. Dale and K. Church, editors, *Proceedings of 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, pages 542–549, Univeristy of Maryland, College Park, Maryland, USA, 1999. Morgan Kaufmann.

C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov Models and their Application to Adaptive Web Navigation. In D. Hand, D. Keim, and R. Ng, editors, *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 143–152, Edmonton, Canada, July 2002. ACM Press.

D. Andre and S. Russell. Programmable Reinforcement Learning Agents. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (NIPS-00)*, pages 1019–1025, Denver, CO, USA, 2001. MIT Press.

D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data. In C. Schmid, S. Soatto, and C. Tomasi, editors, *IEEE Computer Society International Conference on Computer Vision and Pattern Recognition (CVPR-05)*, volume 2, pages 169–176, San Diego, CA, USA, June 20 – 26 2005.

J. K. Baker. Trainable grammars for speech recognition. In J. J. Wolf and D. H. Klatt, editors, *Speech communication papers presented at th 97th Meeting of the Acoustical Society of America*, pages 547–550, Boston, MA, 1979.

O. Bangsø, H. Langseth, and T. D. Nielsen. Structural learning in object oriented domains. In I. Russell and J. Kolen, editors, *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-01)*, pages 340–344, Key West, Florida, USA, 2001. AAAI Press.

E. Bauer, D. Koller, and Y. Singer. Update Rules for Parameter Estimation in Bayesian Networks. In D. Geiger and P. P. Shenoy, editors, *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 3–13, Providence, Rhode Island, USA, 1997. Morgan Kaufmann.

H. Bauer. *Wahrscheinlichkeitstheorie*. Walter de Gruyter, Berlin, New York, 4. edition, 1991.

E. B. Baum. Towards a Model of Intelligence as an Economy of Agents. *Machine Learning Journal*, 35(2):155–185, 1999.

L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

D. P. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

F. Bergadano and D. Gunetti. *Inductive Logic Programming: From Machine Learning to Software Engeneering*. MIT Press, 1996.

J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive Probabilistic Networks with Hidden Variables. *Machine Learning Journal*, 29(2–3):213–244, 1997.

C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

H. Blockeel and L. De Raedt. Lookahead and discretization in ilp. In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh International Workshop on Inductive Logic Programming (ILP-97)*, volume 1297 of *LNCS*, pages 77–85. Springer, 1997.

H. Blockeel and L. De Raedt. Top-down Induction of First-order Logical Decision Trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.

H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Improving the Efficiency of Inductive Logic Programming Through the Use of Query Pack. *Journal of Artificial Intelligence Research (JAIR)*, 16:135–166, 2002.

U. Bohnebeck, T. Horváth, and S. Wrobel. Term comparison in first-order similarity measures. In D. Page, editor, *Proceedings of the Eigth International Conference on Inductive Logic Programming (ILP-98)*, volume 1446 of *LNCS*, pages 65–79, Madison, Wisconsin, USA, July 22–24 1998. Springer.

C. Boutilier, T. Deam, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *JAIR*, 11:1–94, 1999.

C. Boutilier, R. Reiter, and B. Price. Symbolic Dynamic Programming for First-order MDPs. In B. Nebel, editor, *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 690–700, Seattle, USA, 2001. Morgan Kaufmann.

J. A. Boyan and A. W. Moore. Generalization in Reinforcement Learning: Safely Approximating the Value Function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 369–376. The MIT Press, 1995. (Proceedings of NIPS-94, Denver, Colorado, USA, 1994).

J. Bresnan. *Lexical-Functional Syntax.* Blackwell, Malden, MA, 2001.

H. Bui, S. Venkatesh, and G. West. Policy Recognition in the Abstract Hidden Markov Model. *Journal of Artificial Intelligence Research (JAIR)*, 17:451–499, 2002.

W. Buntine. Generalized subumption and its applications to induction and redundancy. *Artificial Intelligence Journal*, 36(2):149–176, 1988.

P. Carbonetto, J. Kisynski, N. de Freitas, and D. Poole. Nonparametric Bayesian Logic. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the Twenty-Firstst Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 85–93, Edinburgh, Scotland, July 26-29 2005.

R. C. Carrasco, J. Oncina, and J. Calera-Rubio. Stochastic inference of regular tree languages. *Machine Learning Journal*, 44(1/2):185–197, 2001.

B. Cestnik. Estimating probabilities: A crucial task in machine learning. In L. Aiello, editor, *Proceedings of the Ninth European Conference on Artificial Iintelligence (ECAI-90)*, pages 147–149, Stockholm, Sweden, 1990. Pitmann Publishing, London/Boston.

S. Chakrabarti, B. Dom, and P. Indyk. Enhanced Hypertext Categorization Using Hyperlink. In L. M. Haas and A. Tiwary, editors, *Proceedings of the ACM International Conference on Management of Data (ACM-SIGMOD-98)*, pages 307–318. ACM Press, June 2-4, 1998.

J. M. Chandonia, G. Hon, N. S. Walker, L. Lo Conte, P.Koehl, and S. E. Brenner. The ASTRAL compendium in 2004. *Nucleic Acids Research*, 32:D189–D192, 2004.

J. Cheng, C. Hatzis, M.–A. Krogel, S. Morishita, D. Page, and J. Sese. KDD Cup 2002 Report. *SIGKDD Explorations*, 3(2):47 – 64, 2002.

A. Cocora, K. Kersting, C. Plagemann, W. Burgard, and L. De Raedt. Learning Relational Navigation Policies. Submitted, 2005.

W. W. Cohen. Grammatically Biased Learning: Learning Logic Programs Using an Explicit Antecedent Description Language. *Artificial Intelligence Journal*, 68:303–366, August 1994.

M. Collins and N. Duffy. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 625–632, Vancouver, British Columbia, Canada, December 3-8 2001. The MIT Press.

G. F. Cooper. The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks. *Artificial Intelligence Journal*, 42:393–405, 1990.

R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic networks and expert systems*. Statistics for engineering and information. Springer-Verlag, 1999.

M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. M. Mitchell, K. Nigam, and S. Slattery. Learning to Construct Knowledge Bases from the World Wide Web. *Artificial Intelligence Journal*, 118(1–2):69–113, 2000.

T. Croonenborghs, J. Ramon, and M. Bruynooghe. Towards Informed Reinforcement Learning. In P. Tadepalli, R. Givan, and K. Driessen, editors, *Working Notes of the ICML-2004 Workshop on Relational Reinforcement Learning*, Banff, Canada, July 8 2004.

J. Cussens. Loglinear models for first-order probabilistic reasoning. In K. Blackmond Laskey and H. Prade, editors, *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 126–133, Stockholm, Sweden, 1999. Morgan Kaufmann.

J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning Journal*, 44(3):245–271, 2001.

J. Cussens. Integrating by separating: Combining probability and logic with ICL, PRISM and SLPs. Technical report, APrIL Projetc, January 2005.

J. Dausset, H. Can, D. Cohen, M. Lthrop, J.-M. Lalouel, and R. White. Centre d'Etude du Polymorphisme humain (CEPH): Collaborative genetic mapping of the human genome. *Genomics*, 6:575–577, 1990.

J. Davis, E. Burnside, I. Dutra, D. Page, and V. Santos Costa. An Integrated Approach to Learning Bayesian Networks of Rules. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *Proceedings of the Sixteenth European Conference of Machine Learning (ECML-05)*, volume 3720 of *LNCS*, pages 84–95, Proto, Portugal, October 3-7 2005.

B. D. Davison and H. Hirsh. Predicting Sequences of User Actions. In A. Danyluk, T. Fawcett, and F. Provost, editors, *Working Notes (WS-98-07) of the AAAI-98/ICML-98 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*, pages 5–12, Madison, WI, USA, July 27 1998. AAAI Press.

L. De Raedt. Logical settings for concept-learning. *Artificial Intelligence Journal*, 95 (1):197–201, 1997.

L. De Raedt and M. Bruynooghe. A Theory of Clausal Discovery. In R. Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1058–1063, Chambery, France, August 28 - September 3 1993. Morgan Kaufmann.

L. De Raedt and L. Dehaspe. Clausal Discovery. *Machine Learning Journal*, 26(2-3): 99–146, 1997.

L. De Raedt and S. Džeroski. First-Order jk-Clausal Theories are PAC-Learnable. *Artificial Intelligence Journal*, 70(1-2):375–392, 1994.

L. De Raedt and K. Kersting. Probabilistic Logic Learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, 5(1):31–48, 2003.

L. De Raedt and K. Kersting. Probabilistic Inductive Logic Programming. In S. Ben-David, J. Case, and A. Maruoka, editors, *Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT-04)*, volume 3244 of *LNCS*, pages 19–36, Padova, Italy, October 2–5 2004. Springer.

L. De Raedt, K. Kersting, and S. Torge. Towards learning stochastic logic programs from proof-banks. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 752–757, Pittsburgh, Pennsylvania, USA, July 9–13 2005. AAAI.

L. De Raedt, N. Lavrač, and S. Džeroski. Multiple Predicate Learning. In R. Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1037–1043, Chambery, France, August 28 - September 3 1993. Morgan Kaufmann.

R. de Salvo Braz, E. Amir, and D. Roth. Lifted First-order Probabilistic Inference. In F. Giunchiglia and L. Pack Kaelbling, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1319–1125, Edinburgh, Scotland, July 30 – August 5 2005. AAAI Press.

T. Dean and K. Kanazawa. Probabilistic Temporal Reasoning. In T. M. Mitchell and R. G. Smith, editors, *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 524–528, St. Paul, MN, USA, August 21-26 1988. AAAI Press / The MIT Press.

R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence Journal*, 89(1):219–283, 1997.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39:1–39, 1977.

U. Dick and K. Kersting. Fisher Kernels for Relational Data. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proceedings of the 15th European Conference on Machine Learning (ECML-06)*, volume 4212 of *LNAI*, Berlin, Germany, September 18–22 2006. Springer. (To appear).

T. G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13:227–303, 2000.

T. G. Dietterich and N. S. Flann. Explanation-based learning and reinforcement learning: a unified view. *Machine Learning Journal*, 28:169–210, 1997.

P. Domingos and M. Richardson. Markov Logic: A Unifying Framework for Statistical Relational Learning. In T. G. Dietterich, L. Getoor, and K. Murphy, editors,

*Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields (SRL-04)*, pages 49–54, Banff, Alberta, Canada, July 8 2004.

K. Driessens. *Relational Reinforcement Learning.* PhD thesis, Department of Computer Science, K.U. Leuven, Leuven, Belgium, May 2004.

K. Driessens, A. Fern, and M. Van Otterlo, editors. *Working Notes of the ICML-2005 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, August 7th 2005.

K. Driessens and J. Ramon. Relational Instance based Regression for Relational Reinforcement Learning. In T. Fawcett and N. Mishra, editors, *Proceedings of the International Conference on Machine Learning (ICML-03)*, pages 123–130, Washington, DC USA, August 21-24 2003.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

S. Džeroski and N. Lavrač, editors. *Relational data mining.* Springer-Verlag, Berlin, 2001.

S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning Journal*, 43(1/2):7–52, 2001.

S. R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994.

A. Eisele. Towards Probabilistic Extensions of Contraint-based Grammars. In J. Dörne, editor, *Computational Aspects of Constraint-Based Linguistics Decription-II.* DYNA-2 deliverable R1.2.B, 1994.

G. Elidan and N. Friedman. Learning the Dimensionality of Hidden Variables. In J. S. Breese and D. Koller, editors, *Proceedings of the Seventeenth Anual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 144–151, Seattle, Washington, USA, 2001. Morgan Kaufmann.

T. Ellman. Explanation-based Learning: A Survey ofPrograms and Perspectives. *ACM COmputing Surveys*, 21(2):163–221, June 1989.

G. W. Euler and H. D. Robertson, editors. *National ITS Program Plan*, volume 1. ITS Joint Program Office and ITS America, 1995.

A. Fern, S. Yoon, and R. Givan. Approximate Policy Iteration with a Policy Language Bias. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16, Vancouver, Canada, 2004. The MIT Press. (Proceedings of NIPS-03,December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada).

D. Fierens, H. Blockeel, M. Bruynooghe, and J. Ramon. Logical Bayesian Networks and Their Relation to Other Probabilistic Logical Models. In S. Kramer and B. Pfahringer, editors, *Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-05)*, volume 3625 / 2005 of *LNCS*, pages 121–135, Bonn, Germany, August 10-13 2005. Springer.

S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden Markov model: analysis and applications. *Machine Learning Journal*, 32:41–62, 1998.

S. Finney, N. H. Gardiol, L. P. Kaelbling, and T. Oates. The Thing That We Tried Didn't Work Very Well: Deictic Representation in Reinforcement Learning. In A. Darwiche and N. Friedman, editors, *Proceedings of the Eighteenth International*

*Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 154–161, Edmonton, Alberta, Canada, August 1-4 2002.

J. Fischer. Asynchronous Relational Value Iteration. Master's thesis, Institute for Computer Science Department, University of Freiburg, 2005.

J. Fischer and K. Kersting. Scaled CGEM: A Fast Accelerated EM. In N. Lavrac, D. Gamberger, H. Blockeel, and L. Todorovski, editors, *Proceedings of the Fourteenth European Conference on Machine Learning (ECML-03)*, volume 2837 of *LNCS*, pages 133–144, Cavtat, Croatia, September 22–26 2003. Springer.

P. Flach. *Simply Logical: Intelligent Reasoning by Example*. John Wiley, 1994.

P. A. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, volume 1634 of *LNAI*, pages 92–103, Bled, Slovenia, 1999. Springer.

P. A. Flach and N. Lachiche. Naive Bayesian classification of structured data. *Machine Learning Journal*, 57(3):233–269, 2004.

R. Forsyth, editor. *Expert Systems: Principles and Case Studies*. Chapman And Hall Computing Series, 1994.

P. Frasconi, A. Passerini, S. H. Muggleton, and H. Lodhi. Declarative kernels. Submitted, 2005.

P. Frasconi, G. Soda, and A. Vullo. Hidden Markov Models for Text Categorization in Multi-Page Documents. *Journal of Intelligent Information Systems*, 18:195–217, 2002.

N. Friedman. Learning Belief Networks in the Presence of Missing Values and Hidden Variables. In D. H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-97)*, pages 125–133, Nashville, TN, USA, July 8-12 1997. Morgan Kaufmann.

N. Friedman. The Bayesian Structural EM Algorithm. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 129–138, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1300–1307, Stockholm, Sweden, July 31 - August 6 1999. Morgan Kaufmann.

B. Fristedt and L. Gray. *A Modern Approach to Probability Theory*. Probability and its applications. Birkhäuser Boston, 1997.

T. W. Frühwirth. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1–3):95–138, 1998.

J. Fürnkranz. Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

J. Fürnkranz. Round Robin Classification. *Journal of Machine Learning Research (JMLR)*, 2:721–747, 2002.

T. Gärtner. Kernel-based Learning in Multi-Relational Data Mining. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, 5(1):49–58, 2003.

T. Gärtner. *Kernels for Structured Data*. PhD thesis, Department of Computer Science, University of Bonn, Bonn, Germany, December 2005.

T. Gärtner, K. Driessens, and J. Ramon. Graph Kernels and Gaussian Processes for Relational Reinforcement Learning. In T. Horváth, editor, *Proceedings of the Thirteenth Internationcal Conference on Inductive Logic Programming (ILP-03)*, volume 2835 of *LNCS*, pages 146–163, Szeged, Hungary, September 29 – October 1 2003. Springer.

T. Gärtner, P. A. Flach, and S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In B. Schölkopf and M. K. Warmuth, editors, *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop (COLT/Kernel-03)*, volume 2777 of *LNCS*, pages 129–143, Washington, DC, USA, August 24-27 2003. Springer.

T. Gärtner, J. W. Lloyd, and P. A. Flach. Kernels and Distances for Structured Data. *Machine Learning Journal*, 57(3):205 – 232, 2004.

S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

L. Getoor. *Learning Statistical Models from Relational Data*. PhD thesis, Stanford University, USA, June 2001.

L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*. Springer, 2001.

L. Getoor, N.Friedman, D. Koller, and B Taskar. Learning Probabilistic Models of Link Structure. *Journal of Machine Leaning Research (JMLR)*, 3:679 – 707, 2002.

Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning Journal*, 29:245–273, 1997.

W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1):169–177, 1994.

R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence Journal*, 147:163–224, 2003.

J. Goodman. Probabilistic feature grammars. In H. C. Bunt, editor, *Proceedings of the Fifth International Workshop on Parsing Technologies (IWPT-97)*, Boston, MA, USA, September 17-20 1997.

G. J. Gordon. Stable fitted reinforcement learning. In *Advances in Neural Information Processing*, pages 1052–1058. MIT Press, 1996.

S. Greenberg. Using Unix: collected traces of 168 users. Technical report, Deptartment of Computer Science, University of Calgary, Alberta, 1988.

A. Großmann, S. Hölldobler, and O. Skvortsova. Symbolic Dynamic Pprogramming within the Fluent Calculus. In N. Ishii, editor, *Proceedings of the IASTED International Conference on Artificial and Computational Intelligence*, pages 378–383, Tokyo, Japan, September 25-27 2002. ACTA Press.

C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing Plans to New Environments in Relational MDPs. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1003–1010, Acapulco, Mexico, August 9-15 2003. Morgan Kaufmann.

B. Gutmann. Relational conditional random fields. Master's thesis, Institute for Computer Science, University of Freiburg, 2005. (In German).

P. Haddawy. Generating Bayesian networks from probabilistic logic knowledge bases. In R. Lopez de Mantaras and D. Poole, editors, *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 262–269, Seattle, Washington, USA, July 29-31 1994.

S. Hanks and D. V. McDermott. Modelling a dynamic and uncertain world I: Symbolic and probabilisitic reasoning about change. *Artificial Intelligence Journal*, 66(1):1–55, 1994.

D. Haussler. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

D. Heckerman. A Tutorial on Learning with Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research,, 1995.

D. Heckerman and J. Breese. Causal Independence for Probability Assessment and Inference Using Bayesian Networks. Technical Report MSR-TR-94-08, Microsoft Research, 1994.

D. Heckerman, D. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering and data visualization. *Journal of Machine Learning Research (JMLR)*, 1:29–75, 2000.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning Journal*, 20(3): 197–243, 1995a.

D. Heckerman, A. Mamdani, and M. P. Wellman. Real-World Applications of Bayesian Networks. *Communications of the ACM*, 38(3):24–26, March 1995b.

N. Helft. Induction as nonmonotonic inference. In R. J. Brachman and H. J. Levesque, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning(KR-89)*, pages 149–156, Toronto, Canada, May 15-18 1989. Kaufmann.

J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

T. Horváth, S. Wrobel, and U. Bohnebeck. Relational Instance-Based learning with Lists and Terms. *Machine Learning Journal*, 43(1/2):53–80, 2001.

T. Hubbard, A. Murzin, S. Brenner, and C. Chotia. *SCOP*: a structural classification of proteins database. *NAR*, 27(1):236–239, 1997.

T. Jaakkola, M. Diekhans, and D. Haussler. Using the Fisher Kernel Method to Detect Remote Protein Homologies. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H. -W. Mewes, and R. Zimmer, editors, *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB-99)*, pages 149–158, Heidelberg, Germany, August 6–10 1999. AAAI Press.

T. Jaakkola and D. Haussler. Exploiting Generative Models in Discriminative Classifiers. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 487–493. The MIT Press, 1999. (Proceedings of NIPS-98, November 30 - December 5, 1998, Denver, Colorado, USA).

N. Jacobs and H. Blockeel. The Learning Shell: Automated Macro Construction. In M. Bauer, P. J. Gmytrasiewicz, and J. Vassileva, editors, *Proceedings of the International Conference on User Modeling (UM-01)*, LNCS, pages 34–43, Sonthofen, Germany, July 13-17 2001. Springer.

N. Jacobs and H. Blockeel. User modeling with sequential data. In C. Stephanidis, editor, *Proceedings of 10th International Conference on Human - Computer Interaction (HCI-03)*, volume 4, pages 557–561, 2003.

M. Jäger. Relational Bayesian Networks. In K. B. Laskey and H. Prade, editors, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 266–273, Stockholm, Sweden, July 30–August 1 1997. Morgan Kaufmann.

M. Jäger. Importance Sampling on Relational Bayesin Networks. Aalborg University, 2005.

M. Jamshidian and R. I. Jennrich. Accleration of the EM Algorithm by using Quasi-Newton Methods. *Journal of the Royal Statistical Society B*, 59(3):569–587, 1997.

F. V. Jensen. *An Introduction to Bayesian Networks*. UCL Press Limited, 1996. Reprinted 1998.

F. V. Jensen. Gradient Descent Training of Bayesian Networks. In A. Hunter and S. Parsons, editors, *Proceedings of the Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-99)*, volume 1638 of *LNCS*, pages 190–200. Springer, 1999.

F. V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag New, 2001.

T. Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Kluwer, 2002.

J. Johns and S. Mahadevan. A Variational Learning Algorithm for the Abstract Hidden Markov Model. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 9–14, Pittsburgh, Pennsylvania, USA, July 9–13 2005. AAAI.

M. Johnson. Learning and parsing stochastic unification-based grammars. In B. Schölkopf and M. K. Warmuth, editors, *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory (COLT-03)*, volume 2777 of *LNCS*, pages 671–683. Springer, 2003.

M. I. Jordan, editor. *Learning Graphical Models*. MIT Press, 1999.

L. P. Kaelbling, T. Oates, N. H. Gardiol, and S. Finney. Learning in worlds with objects. In P. R. Cohen and T. Oates, editors, *Working Notes (SS-01-05) of the AAAI Stanford Spring Symposium on Learning Grounded Representations*, pages 31–36, Stanford, CA, USA, March 26-28 2001. AAAI Press.

Y. Kameya, T. Sato, and N.-G. Zhou. Yet more efficient EM learning for parameterized logic programs by inter-goal sharing. In R. Lopez de Mantaras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, pages 490–494, Valencia, Spain, August 22-27 2004. IOS Press.

E. Karabaev and O. Skvortsova. A Heuristic Search Algorithm for Solving First-Order MDPs. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the 21th International Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 292–299, Edinburgh, Scotland, July 2005. AUAI Press.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized Kernels between Labeled Graphs. In T. Fawcett and N. Mishra, editors, *Proceedings of the International Conference on Machine Learning (ICML-03)*, pages 321–328, Washington, DC USA, August 21-24 2003.

S. M. Katz. Estimation of probabilities from sparse data for hte language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing (ASSP)*, 35:400–401, 1987.

K. Kersting. Bayes'sche-logische Programme. Master's thesis, Institute for Computer Science, University of Freiburg, 2000.

K. Kersting and L. De Raedt. Adaptive Bayesian Logic Programs. In C. Rouveirol and M. Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, volume 2157 of *LNAI*, pages 118–131, Strasbourg, France, September 2001a. Springer.

K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Freiburg, Germany, April 2001b.

K. Kersting and L. De Raedt. Towards Combining Inductive Logic Programming with Bayesian Networks. In Céline Rouveirol and M. Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, volume 2157 of *LNAI*, pages 118–131, Strasbourg, France, September 2001c. Springer.

K. Kersting and L. De Raedt. Basic principles of learning bayesian logic programs. Technical Report 174, Institute for Computer Science, University of Freiburg, Freiburg, Germany, June 2002.

K. Kersting and L. De Raedt. Logical Markov Decision Programs. In L. Getoor and D. Jensen, editors, *Working Notes of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data (SRL-03)*, pages 63–70, Acapulco, Mexico, August 11 2003.

K. Kersting and L. De Raedt. Logical Markov Decision Programs and the Convergence of Logical TD($\lambda$). In A. Srinivasan, R. King, and R. Camacho, editors, *Proceedings of the Fourteenth International Conference on Inductive Logic Programming (ILP-04*, number 3194 in LNCS, pages 180–197, Porto, Portugal, September 6-8 2004. Springer.

K. Kersting and L. De Raedt. Bayesian Logic Programming: Theory and Tool. In L. Getoor and B. Taskar, editors, *An Introduction to Statistical Relational Learning*. MIT Press, 2005. (to appear).

K. Kersting, L. De Raedt, and S. Kramer. Interpreting Baysian Logic Programs. In L. Getoor and D. Jensen, editors, *Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data (SRL-00)*, Austin, Texas, USA,, July 31 2000. AAAI Press.

K. Kersting, L. De Raedt, and T. Raiko. Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR)*, 25:425–456, 2006.

K. Kersting and U. Dick. Balios - The Engine for Bayesian Logic Programs. In J.-F. Boulicaut, F. Esposito anf F. Giannotti, and D. Pedreschi, editors, *Proceedings of the 8th European Conference on Principles and Practice of Knowledege Discovery in Databases (PKDD-04)*, pages 549–551, Pisa, Italy, September 20-25 2004.

K. Kersting and T. Gärtner. Fisher Kernels and Logical Sequences with an Application to Protein Fold Recognition. In C. Campbell and P. Long, editors, *Working Notes of the NIPS-2002 Workshop on Machine Learning Techniques for Bioinformatics*, Vancouver, Canada, December (Friday) 13 2002.

K. Kersting and T. Gärtner. Fisher Kernels for Logical Sequences. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Proceedings of the 15th Euro-*

*pean Conference on Machine Learning (ECML-04)*, volume 3201 of *LNCS*, pages 205 – 216, Pisa, Italy, 2004.

K. Kersting and T. Raiko. 'Say EM' for Selecting Probabilistic Models for Logical Sequences. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 300–307, Edinburgh, Scotland, July 26-29 2005.

K. Kersting, T. Raiko, and L. De Raedt. Logical Hidden Markov Models (Extended Abstract). In J. A. Games and A. Salmeron, editors, *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM-02)*, pages 99–107, Cuenca, Spain, November 6–8 2002.

K. Kersting, T. Raiko, and L. De Raedt. A Structural GEM for Learning Logical Hidden Markov Models. In S. Džeroski, L. De Raedt, and S. Wrobel, editors, *Working Notes of the Second KDD-Workshop on Multi-Relational Data Mining (MRDM-03)*, Washington, DC, USA, August 27 2003a.

K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden markov models. In R. B. Altman, A. K. Dunker, L. Hunter, T. A. Jung, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing (PSB-03)*, pages 192–203, Kauai, Hawaii, USA, January 3–7 2003b. World Scientific.

K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes Relational. In R. Greiner and D. Schuurmans, editors, *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*, pages 465–472, Banff, Alberta, Canada, July 4–8 2004.

K.-E. Kim and T. Dean. Solving factored MDPs using non-homogeneous partitions. *Artificial Intelligence*, 147:225–251, 2003.

M. Koivisto, T. Kivioja, H. Mannila, P. Rastas, and E. Ukkonen. Hidden Markov Modelling Techniques for Haplotype Analysis. In S. Ben-David, J. Case, and A. Maruoka, editors, *Proceedings of 15th International Conference on Algorithmic Learning Theory (ALT-04)*, volume 3244 of *LNCS*, pages 37–52. Springer, 2004.

M. Koivisto, M. Perola, T. Varilo, W. Hennah, J. Ekelund, M. Lukk, L. Peltonen, E. Ukkonen, and H. Mannila. An MDL method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In R. B. Altman, A. K. Dunker, L. Hunter, T. A. Jung, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing (PSB-02)*, pages 502–513. World Scientific, 2002.

S. Kok and P. Domingos. Learning the Structure of Markov Logic Networks. In L. De Raedt and S. Wrobel, editors, *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML-05)*, pages 441–448, Bonn, Germany, August 7-11 2005.

D. Koller. Probabilistic relational models. In S. Džeroski and P. Flach, editors, *Proceedings of Ninth International Workshop on Inductive Logic Programming (ILP-99)*, volume 1634 of *LNAI*, pages 3–13, Bled, Slovenia, 1999. Springer.

D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In M. Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1316–1321, Nagoya, Japan, 1997. Morgan Kaufmann.

B. Korvemaker and R. Greiner. Predicting UNIX command files: Adjusting to user patterns. In W. Iba and S. Rogers, editors, *Working Notes of the AAAI-2000 Spring Symposium on Adaptive User Interfaces*, pages 59–64, Stanford, CA, USA, March 20–22 2000.

D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A Generalized Hidden Markov Model for the Recognition of Human Genes in DNA. In D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and R. Smith, editors, *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology (ISMB-96)*, pages 134–142, St. Louis, MO, USA, July 1996. AAAI.

C.-K. Kwoh and D. F. Gillies. Using hidden nodes in Bayesian networks. *Artificial Intelligence Journal*, 88:1–38, 1996.

W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10(4), 1994.

N. Landwehr, K. Kersting, and L. De Raedt. nFOIL: Integrating Naïve Bayes and Foil. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 795–800, Pittsburgh, Pennsylvania, USA, July 9–13 2005. AAAI.

T. Lane. Hidden Markov Models for Human/Computer Interface Modeling. In Åsa Rudström, editor, *Working Notes of the IJCAI-99 Workshop on Learning about Users*, pages 35–44, Stockholm, Sweden, July 1999.

P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1995.

H. Langseth and O. Bangsø. Parameter learning in Object-Oriented Bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32(1/2):221–243, 2001.

K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

K. B Laskey and P. C. G. Costa. Of Starships and Klingons: Bayesian Logic for the 23rd Century. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 346–353, Edinburgh, Scotland, July 26-29 2005.

S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

N. Lavrač and S. Džeroski. *Inductive Logic Programming*. Ellis Horwood, 1994.

R. Lecoeuche. Learning optimal dialogue management rules by using reinforcement learning and inductive logic programming. In K. Knight, editor, *Proceeding of the Second Meeting of North American Chapter of the Association for Computational Linguistics (NAACL-01)*, pages 1 – 7, Pittsburgh, PA, USA, June 2–7 2001.

S. D. Lee and L. De Raedt. Constraint Based Mining of First Order Sequences in SeqLog. In R. Meo, P. L. Lanzi, and M. Klemettine, editors, *Database Support for Data Mining Application*, number 2686 in LNCS, pages 155–176. Springer, July 2004.

L. S. Levy and A. K. Joshi. Skeletal structural descriptions. *Information and Control*, 2(2):192–211, 1978.

L. Liao, D. Fox, and H. Kautz. Location-Based Activity Recognition using Relational Markov Networks. In F. Giunchiglia and L. Pack Kaelbling, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 773–778, Edinburgh, Scotland, July 30,– August, 5 2005. AAAI Press.

B. Limketkai, L. Liao, and D. Fox. Relational Object Maps for Mobile Robots. In F. Giunchiglia and L. Pack Kaelbling, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1471–1476, Edinburgh, Scotland, July 30,– August, 5 2005. AAAI Press.

M. L. Littman and H. L. S. Younes. Introduction to the Probabilistic Planning Track. In M. L.ittman and H. L. S. Younes, editors, *Working Notes of the Probabilistic Planning Track of the International Planning Competition (IPC-04)*, pages 1–2, 2004.

J. W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 2. edition, 1989.

Q. Lu and L. Getoor. Link-based Classification. In T. Fawcett and N. Mishra, editors, *Proceedings of the International Conference on Machine Learning (ICML-03)*, pages 496–503, Washington, DC USA, August 21-24 2003.

D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, 1984.

S. A. Macskassy and F. Provost. Classification in Networked Data: A toolkit and a univariate case study. Technical Report CeDER-04-08, CeDER Working Paper, Stern School of Business, New York University, New York, New York 10012, USA, 2004.

C. H. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, 1999.

M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn treebank: Annotating predicate argument structure. In C. J. Weinstein, editor, *In ARPA Human Language Technology Workshop*, pages 114–119, Plainsboro, NJ, USA, March 8–11 1994.

Mausam and D. Weld. Solving Relational MDPs with First-Order Machine Learning. In M. Pistore, D. E. Smith, and H. Geffner, editors, *Working Notes ICAPS-2003 Workshop on Planning under Uncertainty and Incomplete Information*, Trento, Italy, June 9th 2003.

A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden States.* PhD thesis, Department of Computer Science, University of Rochester, 1995.

G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions.* Wiley, New York, 1997.

B. Milch, B. Marthi, and S. Russell. BLOG: Relational Modeling with Unknown Objects". In T. G. Dietterich, L. Getoor, and K. Murphy, editors, *Working Notes of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields (SRL-04)*, pages 67–73, Banff, Alberta, Canada, July 8 2004.

B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. BLOG: Probabilistic Models with Unknown Objects. In F. Giunchiglia and L. Pack Kaelbling, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1352–1359, Edinburgh, Scotland, July 30,– August, 5 2005. AAAI Press.

T. M. Mitchell. *Machine Learning.* The McGraw-Hill Companies, Inc., 1997.

M. Møller. A Scaled Conjugate Gradient Algoritm for Fast Supervised Learning. *Neural Networks*, 6:525–533, 1993.

R. J. Mooney and J. M. Zelle. Integrating ILP and EBL. *SIGART Bulletin*, 5(1): 12–21, January 1994.

A. W. Moore and M. S. Lee. Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. *Journal of Artificial Intelligence Research (JAIR)*, 8: 67–91, 1998.

S. H. Muggleton. Inverse Entailment and Progol. *New Generation Computing Journal*, pages 245–286, 1995.

S. H. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

S. H. Muggleton. Learning Stochastic Logic Programs. *Electronic Transactions in Artificial Intelligence*, 4(041), 2000a.

S. H. Muggleton. Learning stochastic logic programs. In L. Getoor and D. Jensen, editors, *Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data (SRL-00)*, pages 36–41, Austin, Texas, July 31 2000b. AAAI Press.

S. H. Muggleton. Learning Structure and Parameters of Stochastic Logic Programs. In S. Matwin and C. Sammut, editors, *Proceedgins of the Twelfth International Conference on Inductive Logic Prgramming (ILP-02)*, volume 2583 of *LNCS*, Sydney, Australia, July 9–11 2002. Springer.

S. H. Muggleton and C. Feng. Efficient Induction of Logic Programs. In S. H. Muggleton, editor, *Inductive Logic Programming*. Acadamic Press, 1992.

S. H Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19(20):629–679, 1994.

R. Munos and A. Moore. Influence and Variance of a Markov Chain : Application to Adaptive Discretization in Optimal Control. In *Proceedings of the 38th IEEE Conference on Decision and Control (CDC-99)*, volume 2, pages 1464 – 1469, December 7–10 1999.

K. P. Murphy and M. A. Paskin. Linear Time Inference in Hierarchical HMMs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 833–840. The MIT Press, 2002. (Proceedings of NIPS-01, Vancouver, British Columbia, Canada, December 3-8, 2001).

D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. J. Paul. The Challenges of Real-Time AI. *IEEE Computer*, 28(1):58–66, 1995.

C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative Bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.

J. Neville and D. Jensen. Iterative Classification in Relational Data. In L. Getoor and D. Jensen, editors, *Working Notes (WS-00-06) of the AAAI Workshop on Learning Statistical Models From Relational Data (SRL-00)*, pages 42–49, Austin, Texas, July 31 2000. AAAI Press.

J. Neville and D. Jensen. Dependency Networks for Relational Data. In R. Rastogi, K. Morik, M. Bramer, and X. Wu, editors, *Proceedings of The Fourth IEEE International Conference on Data Mining (ICDM-04)*, pages 170–177, Brighton, UK, November 1–4 2004. IEEE Computer Society Press.

J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning Relational Probability Trees. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, editors, *Proceedings of the 9th ACM-SIGKDD International Conference on Knowledge Discovery and*

*Data Mining (KDD-03)*, pages 635–630, Washington, D.C., USA, August 24 – 27 2003.

R. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

L. Ngo and P. Haddawy. Probabilistic logic programming and Bayesian networks. In K. Kanchanasut and J.-J. Levy, editors, *Proceedings of the Asian Computing Science Conference on Algorithms, Concurrency and Knowledge (ACSC-95)*, volume 1023 of *LNCS*, pages 286–300, Pathumthai, Thailand, December 11-13 1995. Springer.

L. Ngo and P. Haddawy. Answering Queries from Context-Sensitive Probabilistic Knowledge Bases. *Theoretical Computer Science*, 171:147–177, 1997.

S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. Springer, 1997.

L. E. Ortiz and L. P. Kaelbling. Accelerating EM: An Empirical Study. In K. B. Laskey and H. Prade, editors, *Proceedings of the Fifteenth Annual Conference on Uncertainty in Articial Intelligence (UAI-99)*, pages 512–521, Stockholm, Sweden, 1999a. Morgan Kaufmann.

L. E. Ortiz and L. P. Kaelbling. Notes on methods based on maximum-likelihodd estimation for learning the parameters of the mixture-of-Gaussians model. Technical Report CS-99-03, Department of Computer Science, Brown University, 1999b.

M. Ostendorf and H. Singer. HMM topology design using maximum likelihood successive state splitting. *Computer Speech and Language*, 11(1):17–41, 1997.

A. Paes, K. Revoredo, G. Zaverucha, and V. Santos Costa. Probabilistic First-Order Theory Revision from Examples. In S. Kramer and B. Pfahringer, editors, *Proceedings of 15th International Conference on Inductive Logic Programming (ILP-05)*, volume 3625 / 2005 of *LNCS*, Bonn, Germany, August 10-13 2005. Springer.

A. Passerini, P. Frasconi, and L. De Raedt. Kernels on Prolog Proof Trees: Statistical Learning in the ILP Setting. In E. Kitzelmann, R. Olsson, and U. Schmid, editors, *Working Notes of the ICML '05 Workshop on Approaches and Applications of Inductive Programming (AAIP-05)*, Bonn, Germany, August 7th 2005.

J. Pearl. *Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2. edition, 1991.

A. Pfeffer. IBAL: A Probabilistic Rational Programming Language. In B. Nebel, editor, *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 733–740, Seattle, Washington, USA, 2001. Morgan Kaufmann.

A. J. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Computer Science Department, Stanford University, December 2000.

J. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1998.

G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, 1970.

C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, 1994.

D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence Journal*, 64:81–129, 1993.

D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1–2):7–56, 1997.

D. Poole. First-order probabilistic inference. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 985–991, Acapulco, Mexico, August 2003. Morgan Kaufmann.

A. Puech and S. H. Muggleton. A comparison of stochastic logic programs and bayesian logic programs. In L. Getoor and D. Jensen, editors, *Working Notes of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data (SRL-03)*, pages 121–129, Acapulco, Mexico, August 11 2003.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

J. R. Quinlan. Determinate literals in inductive logic programming. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 746–750, Sydney, Australia, 1991. Morgan Kaufmann.

J. R. Quinlan and R. M. Cameron-Jones. Induction of logic programs:FOIL and related systems. *New Generation Computing*, pages 287–312, 1995.

L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

L. R. Rabiner and B. H. Juang. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

T. Raiko, K. Kersting, J. Karhunen, and L. De Raedt. Bayesian Learning of Logical Hidden Markov Models. In *Proceedings of the Finnish AI conference (STeP-02)*, pages 64–71, Oulu, Finland, December 15–17 2002.

J. Ramon. On the convergence of relational reinforcement learning using a decision tree learner. In K. Driessens, A. Fern, and M. Van Otterlo, editors, *Working Notes of the ICML-2005 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, August 7th 2005.

J. Rennie and A. McCallum. Using Reinforcement Learning to Spider the Web Efficiently. In I. Bratko and S. Džeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, Bled, Slovenia, June 27 – 30 1999. Morgan Kaufmann.

P. Resnick. SocioTechnical Support for Ride Sharing. In *Working Notes of the Symposium on Crossing Disciplinary Boundaries in the Urban and Regional Contex (UTEP-03)*, Ann Arbor, MI, USA, March 7,21,28, April 4 2003. Rackham School of Graduate Studies, University of Michigan.

K. Revoredo and G. Zaverucha. Revision of first-order Bayesian classifiers. In S. Matwin and C. Sammut, editors, *In Proceedings of the 12th International Conference on Inductive Logic Programming (ILP-02)*, volume 2583 of *LNCS*, pages 223–237, Sydney, Australia, July 9–11 2002. Springer.

M. Richardson and P. Domingos. Markov Logic Networks. *Submitted*, 2005.

S. Riezler. Statistical Inference and Probabilistic Modelling for Constraint-Based NLP. In B. Schrder, W. Lenders, and W. Hess und T. Portele, editors, *Proceedings of the 4th Conference on Natural Language Processing (KONVENS-98)*, 1998. Also as CoRR cs.CL/9905010.

F. Riguzzi. Learning logic programs with annotated disjunctions. In A. Srinivasan, R. King, and R. Camacho, editors, *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP-04)*, volume 3194 of *LNCS*, pages 270–287, Porto, Portugal, September 6-8 2004. Springer.

S. Rosweis and Z. Ghahramani. A Uunifying Review of Linear Gaussian Models. *Neural Computation*, 11(2):305–345, 1999.

D. Roth. On the Hardness of Approcimate Reasoning. *Artificial Intelligence Journal*, 82:273–302, 1998.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., 1995.

Y. Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60, 1992.

Y. Sakakibara. Pair hidden markov models on tree structures. *Bioinformatics*, 19 (Suppl.1):i232–i240, 2003.

Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjolander, and R. Underwood. Stochastic context-free grammars for tRNA modelling. *Nucleic Acids Research*, 22 (23):5112–5120, 1994.

R. Salakhutdimov, S. Roweis, and Z. Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. In T. Fawcett and N. Mishra, editors, *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, Washington, DC USA, August 21-24 2003.

S. Sanghai, P. Domingos, and D. Weld. Dynamic probabilistic relational models. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03 )*, pages 992–997, Acapulco, Mexico, August 2003. Morgan Kaufmann.

S. Sanghai, P. Domingos, and D. Weld. Learning Models of Relational Stochastic Processes. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *Proceedings of the Sixteenth European Conference on Machine Learning (ECML-05)*, volume 3720 of *LNCS*, pages 715–723, Porto, Portugal, October 3-7 2005. Springer.

S. Sanner and C. Boutilier. First-order algebraic decision diagrams (FOADDs) and their application to symbolic dynamic programming. Unpublished manuscript, 2004.

S. Sanner and C. Boutilier. Approximate Linear Programming in First-Order MDPs. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 509–517, Edinburgh, Scotland, July 26-29 2005.

S. Sanner and D. McAllester. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In F. Giunchiglia and L. Pack Kaelbling, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1384–1391. AAAI Press, July 30,– August, 5 2005.

V. Santos Costa, D. Page, M. Qazi, and J. Cussens. CLP(BN): Constraint logic programming for probabilistic knowledge. In U. Kjaerulff and C. Meek, editors, *Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 517–524, 2003a.

V. Santos Costa, A. Srinivasan, R. Camacho, H. Blockeel, B. Demoen, G. Janssens, J. Struyf, H. Vandecasteele, and W. Van Laer. Query Transformations for Improving the Efficiency of ILP Systems. *Journal of Machine Learning Research (JMLR)*, 4:465–491, 2003b.

T. Sato. A Statistical Learning Method for Logic Programs with Distribution Semantics. In L. Sterling, editor, *Proceedings of the Twelfth International Conference on Logic Programming (ICLP-95)*, pages 715 – 729, Tokyo, Japan, 1995. MIT Press.

T. Sato and Y. Kameya. Parameter Learning of Logic Programs for Symbolic-Statistical Modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454, 2001.

T. Sato and Y.Kameya. A dynamic programming approach to parameter learning of generative models with failure. In T. G. Dietterich, L. Getoor, and K. Murphy, editors, *Working Notes of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields (SRL-04)*, pages 94–101, Banff, Alberta, Canada, July 8 2004.

C. Saunders, J. Shawe-Taylor, and A. Vinokourov. String kernels, fisher kernels and finite state automata. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 633–640. The MIT Press, 2003. (Proceedings of NIPS-02, December 9-14, 2002, Vancouver, British Columbia, Canada ).

R. D. Schachter. Bayes-Ball: The Rational Pasttime (for Determining Irrelevance and Requisite Information in Belief Networks and Influence Diagrams). In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 480–487, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.

G. Shani, D. Heckerman, and R. I. Brafman. An MDP-Based Recommender System. *Journal of Machine Learning Research (JMLR)*, pages 1265–1295, 2005.

E. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.

J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

Y.-D. Shen and Q. Yang. Deriving a Stationary Dynamic Bayesian Network from a Logic Program with Recursive Loops. In S. Kramer and B. Pfahringer, editors, *Proceedings of 15th International Conference on Inductive Logic Programming (ILP-05)*, volume 3625 / 2005 of *LNCS*, Bonn, Germany, August 10-13 2005. Springer.

S. M. Siddiqi and A. W. Moore. Fast Inference and Learning in Large-State-Space HMMs. In L. De Raedt and S. Wrobel, editors, *Proceedings of the Twenty Second International Conference on Machine Learning (ICML-05)*, pages 801–808, Bonn, Germany, August 7-10 2005.

S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. S. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing*, volume 7, pages 361–268. The MIT Press, 1995. (Proceedings of NIPS-94, Denver, Colorado, USA).

P. Singla and P. Domingos. Discriminative training of markov logic networks. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 868–873, Pittsburgh, Pennsylvania, USA, July 9–13 2005. AAAI Press.

J. Slaney and S. Thiébaux. Blocks World revisited. *Artificial Intelligence Journal*, 125:119–153, 2001.

N. Smith and M. Gales. Speech recognition using SVMs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 1197–1204. The MIT Press, 2002. (Proceedings of NIPS-01, Vancouver, British Columbia, Canada, December 3-8, 2001).

A. Srinivasan. *The Aleph Manual*, 1999. Available at `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`.

A. Srinivasan, S. H. Muggleton, and M. Bain. The Justification of Logical Theories Based on Data Compression. In K. Furukawa, D. Michie, and S. H. Muggleton, editors, *Machine Intelligence 13*, pages 87–121, Strathclyde University's Ross Priory retreat on Loch Lomond,Scotland, January 1994. Oxford University Press.

A. Srinivasan, S. H. Muggleton, R. D. King, and M. J. E. Sternberg. Theories for Mutagenicity: A Study of First-Order and Feature -based Induction. *Artificial Intelligence Journal*, 85:277–299, 1996.

I. Stahl. Predicate Invention in ILP — an Ooverview. In P. Brazdil, editor, *Proceedings of the Sixth European Conference on Machine Learning (ECML-93)*, volume 667, pages 313–322, Vienna, Austria, April 5–7 1993. Springer.

L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, 1986.

A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18. Morgan Kaufman, 1993. (Proceedings of NIPS-92, Denver, Colorado, USA, November 30 - December 3, 1992).

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063. The MIT Press, 2000. (Proceedings NIPS-99, Denver, Colorado, USA, November 29 - December 4, 1999).

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence Journal*, 112:181–211, 1999.

P. Tadepalli, R. Givan, and K. Driessen, editors. *Working Notes of the ICML-2004 Workshop on Relational Reinforcement Learning*, Banff, Canada, July 8 2004.

B. Taskar, P. Abbeel, and D. Koller. Discriminative Probabilistic Models for Relational Data. In A. Darwiche and N. Friedman, editors, *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 485–492, Edmonton, Alberta, Canada, August 1-4 2002.

B. Taskar, V. Chatalbashev, and D. Koller. Learning Associative Markov Networks. In R. Greiner and D. Schuurmans, editors, *Proceedings of the Twenty-first International Conference on Machine Learning (ICML-04)*, pages 102–110, Banff, Alberta, Canada, July 4–8 2004a.

B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning Structured Prediction Models: A Large Margin Approach. In L. De Raedt and S. Wrobel, editors, *Proceedings of the Twenty Second International Conference on Machine Learning (ICML-05)*, pages 897–902, Bonn, Germany, August 7-10 2005.

B. Taskar, C. Guestrin, and D. Koller. Max-Margin Markov Networks. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16, Vancouver, Canada, 2004b. The MIT Press. (Proceedings of NIPS-03, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada).

B. Taskar, E. Segal, and D. Koller. Probabilistic Clustering in Relational Data. In B. Nebel, editor, *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 870–877, Seattle, Washington, USA, 2001. Morgan Kaufmann.

B. Thiesson. Accelerated quantification of Bayesian networks with incomplete data. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of First ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-95*, pages 306–311, Montreol, Canada, 1995. AAAI Press.

I. Thon. Logische Markow Modelle. Institute for Computer Science, University of Freiburg. (In German), 2004.

R. Triebel, K. Kersting, and W. Burgard. Robust 3D Scan Point Classification using Associative Markov Networks. In N. Papanikolopoulos, editor, *IEEE International Conference on Robotics and Automation (ICRA-06)*, Walt Disney World Resort in Orlando, Florida, USA, May 15–19 2006. (To appear).

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions of Automatic Control*, 42:674–690, 1997.

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.-R. Müller. A new discriminative kernel from probabilistic models. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 977–984. The MIT Press, 2002a. (Proceedings of NIPS-01, December 3-8, 2001, Vancouver, British Columbia, Canada).

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 2002b.

M. Turcotte, S. H. Muggleton, and M. J. E. Sternberg. The Effect of Relational Background Knowledge on Learning of Protein Three-Dimensional Fold Signatures. *Machine Learning Journal*, 43(1/2):81–95, 2001.

W. Uwents and H. Blockeel. Classifying Relational Data with Neural Networks. In S. Kramer and B. Pfahringer, editors, *Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-05)*, volume 3625 / 2005 of *LNCS*, pages 384–396, Bonn, Germany, August 10-13 2005. Springer.

L. G. Valiant. A theory of the Learnable. *Communications of the ACM*, 27(11): 1134–1142, November 1984.

W. Van Laer and L. De Raedt. How to Upgrade Propositional Learners to First Order Logic: A Case Study. In N. Lavrač and S. Džeroski, editors, *Relational Data Mining*. Springer Verlag, 2001.

M. Van Otterlo. Reinforcement Learning for Relational MDPs. In A. Nowe, T. Lenaerts, and K. Steehaut, editors, *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands (BeneLearn-04)*, Brussels, Belgium, January 8–9 2004.

J. Vennekens and S. Verbaeten. A General View on Probabilistic Logic Programming. In T. Heskes, P. Lucas, L. Vuurpijl, and W. Wiegerinck, editors, *Proceedings 15th Belgian-Dutch Conference on Artificial Intelligence (BNAIC-03)*, Kasteel Heyendael, Nijmegen, The Netherlands, October 23–24 2003.

J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic Programs with Annotated Disjunctions. In B. Demoen and V. Lifschitz, editors, *Proceedings of 20th International Conference on Logic Programming (ICLP-04)*, pages 431–445, Saint-Malo, France, September 6-10 2004.

C. Watkins. Kernels from matching operations. Technical report, Department of Computer Science, Royal Holloway, University of London, 1999.

S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning Journal*, 7(1):45 – 83, 1991.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning Journal*, 8:229–256, 1992.

R. J. Williams and D. Zipser. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. In *Back-propagation:Theory, Architectures and Applications*, pages 433–486. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1995.

I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

K.-J. Won, A. Prügel-Bennett, and A. Krogh. The Block Hidden Markov Model for Biological Sequence Analysis. In M. G. Negoita, R. J. Howlett, and L. C. Jain, editors, *Proceedings of the Eighth International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES-04)*, volume 3213 of *LNCS*, pages 64–70. Springer, 2004.

S. Wrobel. First Order Theory Refinement. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.

Y. Xiang, S. K. M. Wong, and N. Cercone. Critical Remarks on Single Link Search in Learning Belief Networks. In E. Horvitz and F. V. Jensen, editors, *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 564–571, Portland, Oregon, USA, 1996. Morgan Kaufmann.

J. S. Yedidia, E. T. Freeman, and Y. Weiss. Generalized Belief Propagation. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 689–695, Denver, CO, USA, 2001. The MIT Press.

S. Yoon, A. Fern, and R. Givan. Inductive policy selection for first-order MDPs. In A. Darwiche and N. Friedman, editors, *Proceedings of the Eighteenth International Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 568–576, Edmonton, Alberta, Canada, August 1-4 2002.

L. S. Zettlemoyer, H. M. Pasula, and L. Pack Kaelbling. Learning Planning Rules in Noisy Stochastic World. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 911–918, Pittsburgh, Pennsylvania, USA, July 9–13 2005. AAAI.

# Symbol Index

## Overture

| | |
|---|---|
| $\mathtt{p}/n$ | Predicate symbol $\mathtt{p}$ of arity $n$ 10 |
| $\Sigma$ | First order alphabet 10 |
| $\mathtt{A} :\!-\mathtt{B}_1, \ldots, \mathtt{B}_m$ | Definite clause 10 |
| head($c$) | Head of clause c 10 |
| body($c$) | Body of clause c 10 |
| Var($\cdot$) | Set of all variables in a term, atom, or clause 10 |
| $\theta$ | Substitution 10 |
| mgu($\cdot,\cdot$) | Most general unifier 11 |
| $\preceq$ | $\theta$-subsumption 11 |
| lgg($\cdot,\cdot$) | Least general generalization 11 |
| glb($\cdot,\cdot$) | Greatest lower bound 11 |
| hb($\cdot$) | Herbrand base 11 |
| $A \models B$ | $A$ entails $B$ 11 |
| LH($\cdot$) | Least Herbrand model 11 |
| $T_\cdot$ | Immediate consequence operator 11 |
| $\mathcal{H}$ | Hypothesis space 19 |
| $P$ | Probability value 20 |
| $\mathbf{P}$ | Probability distribution 20 |
| $\mathtt{x}$ | Random variable 20 |
| $x$ | State of random variable 20 |
| $\mathbf{X}$ | Set of random variables 20 |
| x | Set of states of random variables 20 |

## Part I

| | |
|---|---|
| $\mathbf{S}(\cdot)$ | Domain of a random variable 38 |
| $\mathbf{Pa}(\cdot)$ | Parents of a random variable 38 |
| $\mathbf{Fa}(\cdot)$ | Family of a random variable 39 |
| cpd($\mathtt{x}$) | Conditional probability distribution associated with a random variable 39 |
| $\mathtt{A} \mid \mathtt{A}_1, \ldots, \mathtt{A}_n$ | Bayesian clause 40 |
| cpd($c$) | Conditional probability distribution associate with a Bayesian clause 41 |
| cr($\mathtt{p}/l$) | Combining rule 41 |
| $\mathcal{H}$ | Hypothesis space 56 |
| $\mathcal{L}$ | Syntactic bias 56 |
| $L(\mathbf{D}, \boldsymbol{\lambda})$ | Likelihood 63 |
| en($\cdot$) | Expected counts 66 |

## Part II

| | |
|---|---|
| $G_\Sigma(\mathtt{A})$ | set of ground atoms over $\Sigma$ 92 |
| $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ | Abstract transition 92 |
| $\mu$ | Selection distribution 93 |
| $\mathtt{B}-$ parts | Bodies of abstract transitions 94 |
| $p : \mathtt{H} \leftarrow \mathtt{start}$ | probability of starting in state $\mathtt{H}$ 95 |
| $\Upsilon$ | Prior distribution 95 |
| $\Delta$ | set of abstract transitions 95 |
| $(\Sigma, \mu, \Delta, \Upsilon)$ | Logical HMM 95 |
| $T_M$ | Immediate state operator 97 |
| $E_M$ | Immediate state operator 97 |
| $P(\mathtt{O}, \mathtt{S}' \mid \mathtt{S})$ | Transition-emission probability 98 |
| $\boldsymbol{\lambda}$ | Parameters of a logical HMM 101 |
| $\boldsymbol{\lambda}^*$ | Most likely parameters 101 |
| $\alpha_t(\mathtt{S})$ | Forward probability 102 |
| $\beta_t(\mathtt{S})$ | Backward probability 103 |
| $\delta_t(\mathtt{S})$ | highest probability along a single path at time $t$, which accounts for the first $t$ observations and ends in state $\mathtt{S}$ 103 |
| $\xi(\mathtt{T})$ | expected number of times of making the transitions $\mathtt{T}$ at any time 104 |
| $\mathcal{M}$ | Hypothesis space 117 |
| $score_{\mathbf{D}}$ | Scoring function 118 |
| $Pen(\cdot,\cdot,\cdot)$ | Penalty 118 |
| $Q$ | Expected score 120 |

## Intermezzo

| | |
|---|---|
| $U_{\mathbf{x}}$ | Fisher score mapping 137 |

## Part III

# Index

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank