PARSING WITH PRINCIPLES AND CLASSES OF INFORMATION

PARSING WITH PRINCIPLES AND CLASSES OF INFORMATION

by

PAOLA MERLO
University of Geneva,
Switzerland

KLUWER ACADEMIC PUBLISHERS
DORDRECHT / BOSTON / LONDON

A C.I.P. Catalogue record for this book is available from the Library of Congress

ISBN-13: 978-94-010-7265-6 e-ISBN-13: 978-94-009-1708-8
DOI: 10.1007/978-94-009-1708-8

Published by Kluwer Academic Publishers,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

Kluwer Academic Publishers incorporates
the publishing programmes of
D. Reidel, Martinus Nijhoff, Dr W. Junk and MTP Press.

Sold and distributed in the U.S.A. and Canada
by Kluwer Academic Publishers,
101 Philip Drive, Norwell, MA 02061, U.S.A.

In all other countries, sold and distributed
by Kluwer Academic Publishers Group,
P.O. Box 322, 3300 AH Dordrecht, The Netherlands.

Printed on acid-free paper

CONTENTS

PREFACE

The efficient computation of a syntactic representation for a sentence of natural language is a difficult task. Many different types of information must be brought to bear which interact in complex ways. The question then arises of how to separate the different aspects of the computation to make it more efficient and more manageable.

In the actual development of parsers for syntactic analysis, it is standard practice to posit two working levels: the grammar, on one hand, and the algorithms, which produce the analysis of the sentence by using the grammar as the source of syntactic knowledge, on the other hand. Usually, the grammar is derived directly from the work of theoretical linguists. The interest in building a parser which is grounded in a linguistic theory as closely as possible rests on two sets of reasons: first, theories are developed to account for empirical facts about language in a concise way: they seek general. abstract, language-independent explanations for linguistic phenomena; second, current linguistic theories are supposed to be models of humans' knowledge of language. Parsers that can use grammars directly are more likely to have wide coverage, and to be valid for many languages, and they also constitute the most economical model of the human ability to put knowledge of language to use. Therefore, the postulation of a direct correspondence between the parser and theories of grammar is usually assumed as a starting point of investigation.

Experimentation with so-called principle-based architectures has shown that this kind of computation is often inefficient. Inefficiency is a problem that cannot simply be cast aside. Computationally, it renders the use of linguistic theories impractical, and, empirically, it clashes with the observation that humans make use of their knowledge of language very effectively.

In the present work I propose a parsing design that is both computationally and linguistically justified. I start from the observation that linguistic information belongs to five main classes. These classes are defined according to their *information content*, for example topological properties of the tree or lexical information. These classes define the information "modules" of the parser. The data structures and the architecture of the parser mirror the partitioning of linguistic principles according to their information content. Moreover, I observe that linguistic principles are complex constraints. They can be factored into simple constraints, which are precompiled off-line. Thus, I address the issue of how to compute a syntactic representation efficiently by a specific instantiation of partial precompilation.

Computationally, the ensuing organization of the parser is compact and non-redundant: the parser is implemented as an LR parser which is encoded in only a small number of states; information about category and other lexical properties is encoded in a different table which interacts with the LR table on-line. Moreover, a complex phenomenon, long-distance dependencies, is computed efficiently, by making use of information available in the local context of the application of parsing rules. This design can be easily extended to other languages: algorithms are provided for long distance, cyclic movement in Italian and English. Finally, I argue that, psycholinguistically, the proposed design captures some experimental evidence about the interaction of lexical ambiguity with structural ambiguity.

This work contributes some results to the investigation of parsing, both from a computational and from a cognitive point of view. First, a linguistic classification of principles according to their content of information is provided, which is then supported by a a comparison of different compilations of several linguistically-based grammars. It is shown that the grammar built according to the assumptions developed in this work is the most compact and least ambiguous. The proposed distinction between hierarchical and categorial/lexical information is supported by experimental evidence about lexical ambiguity. Furthermore, algorithms for the computation of long distance dependencies are discussed, and compared to other algorithms. Finally, a unified, parameterized algorithm is proposed to treat *wh*-questions in Italian and English, which improves on previous proposals.

This book is a revised version of my 1992 Ph.D. Dissertation. Some of the issues discussed here and some of the results have been previously presented as a student paper at the *30th Annual Meeting of the Association for Computational Linguistics* and are forthcoming as an article in *Computational Linguistics*.

In Chapter 1 I review the debate on the relationship between grammars and parsers, which has animated much recent literature on processing. I propose to tackle the issue as a computational problem and I assume a particular type of partial precompilation of the linguistic principles. The specific hypothesis put forth in this chapter inspires the design of a parser which is presented in detail in the rest of the book.

In Chapter 2 I give an overall view of the organization of the parser and a sample of its parsing capabilities, and I compare it to other parsers in the literature.

In Chapter 3 the parsing engine, an LR(k) parser, is illustrated. I present comparative results on the compilation of grammars according to the LR, LL and LC compilation method. I compare grammar rules that are progressively more distant from bare $\overline{X}$ templates by incrementally adding categorial information. Interestingly, adding categorial information does not appear to reduce the nondeterminism in the grammar. The chapter concludes with some psycholinguistic results to support the approach presented.

The assignment of features is discussed in Chapter 4. I distinguish between features that can be assigned within a maximal projection and features that do not, and I introduce the algorithms to perform local feature annotation, and to compute chains. I discuss the issue of incremental parsing in an LR(k) architecture.

Chapter 5 presents the implementation of the routines that compute locality restrictions on *wh*-movement. It also presents a detailed argumentation in favour of LR(k) parsing for cross-linguistic variation of restrictions on locality, drawing evidence from Italian and English.

This work is the result of my interaction with many people who have helped, encouraged, challenged and criticized me. Their support and their critique have been equally useful, indeed necessary, and I would like to thank them for taking the time to listen.

Special thanks to Amy Weinberg, Eric Wehrli and Uli Frauenfelder for getting me started on this work and for supporting me all along, in many intellectual and practical ways, enabling me to pursue my research interests in a really priviledged environment.

Other people have provided me with insightful comments, at different stages of the work: Michael Brent, Robin Clark, Matthew Crocker, Bonnie Dorr, Paul Gorrell, Sandiway Fong, Bob Frank, Luigi Rizzi, Graham Russell, Suzanne Stevenson.

Many thanks to Celine Courtin for setting up the bibliography.

My colleagues, friends and family will certainly be happy to know I am finally finished working on this document. I would like to thank them for bearing with me. Expecially Suzanne, Graham, Fred, and most of all, my sister Anna.

1 GRAMMARS AND PARSERS

1.1 INTRODUCTION

The efficient understanding of a sentence of natural language is a difficult task, the solution of which calls into play knowledge derived from several disciplines, such as linguistics, computer science, and, often, psychology. One of the principal sources of difficulty in the solution of the problem is constituted by the complex interactions between different types of information, such as lexical, morphological, syntactic, semantic, pragmatics and word knowledge, to which one can add issues of language use, such as frequency of usage and domain-dependent terminolgy.

The question then arises of how to separate the different aspects of computing a representation, to make the computation more efficient and more manageable. There is shared consensus that the different linguistic levels should be tackled by different computational means, or at least, explored independently. Within each level of inquiry the same divide-and-conquer methodology is often adopted, for scientific and engineering reasons.

In the development of parsers for syntactic analysis, it is standard practice to posit two working levels: the grammar, on one hand, and the algorithms, which produce the analysis of the sentence by using the grammar as the source of syntactic knowledge, on the other hand. Usually, the grammar is derived directly from the work of theoretical linguists. The interest in building a parser which is grounded in a linguistic theory as closely as possible rests on two sets of reasons: first, theories are developed to account for empirical facts about language in a concise way: they seek general, abstract, language-independent explanations for linguìstic phenomena; second, current linguistic theories are supposed to be models of humans' knowledge of language. This search for generality is not unique to Government and Binding theory. Feature-structure formalisms also use rule schemata to capture similarities among grammar rules. Reentrancy as a notational device to express common features, moreover, seeks the same type of representational economy which is expressed by the use of "traces" in GB theory.

Parsers that can use grammars directly are more likely to have wide coverage, and to be valid for many languages, and they also constitute the most economical model of the human ability to put knowledge of language to use. Therefore, the postulation of a direct correspondence between the parser and theories of grammar is usually assumed as a starting point of investigation.[1]

Experimentation with so-called principle-based architectures has shown that this kind of computation is often inefficient. Inefficiency is a problem that cannot simply be cast aside. Computationally, it renders the use of linguistic theories impractical, and, empirically, it clashes with the observation that humans make use of their knowledge of language very effectively.

In order to understand what is implied in assuming a direct mapping from the grammar to the parser, as it is currently assumed in principle-based parsing, let's consider first what it means to parse a sentence.

For instance, a property which is clearly important in English is linear order. *John loves Mary* is not equivalent to *Mary loves John*. Also, the word *love* denotes an action, while *John, Mary* denote the participants in the action, the *agent* and the *theme*. We also notice that the notion of participant in an event, or *thematic relation* to a verb, is not the same as that of *grammatical function* such as *subject, object*. A subject agrees in number (plural, singular) with the verb, both when it is an agent or a patient, as (1) shows.

(1) a. John loves the children
b. The children are loved by John.

These are examples of which pieces of information must be recovered. They are implicitly stored in a sentence, according to the rules of a grammar. A

[1]Throughout the work the Government and Binding framework is assumed. I refer the reader to Haegeman (1991). For the reader who is not interested in exploring the details of the theory, a glossary is provided in Appendix A, where I define the terminology used without previous background.

grammar is a finite depository of explicit information, according to which an infinite number of sentences can be constructed.

Information can be stored in a grammar in very different ways. Grammatical theory in the 70s talked about "dative shift", "topicalization", "passive", and it meant that each of these constructions was captured in the grammar by a specific rule. Consequently, rules were not only construction-specific, but also language-specific, (French, Italian and Spanish, for instance, have no "dative shift").

The conceptual development in the 80s, which has given rise to GB theory, consists in having identified the unifying principles of many of these construction specific rules. A new theory is being developed in which a small set of principles applies deductively to generate very different constructions. Thus, for example, all the rules mentioned above included "movement" of some lexical element. Take, for instance, the raising construction, exemplified in (2).

(2) John seems [$_{IP}$t to like Bill]

The structure in (2) respects the linear order given by the English string "John seems to like Bill", and it has the meaning of "It seems that John likes Bill". *John* is both the subject of *seem* and *like*. Thus *John* acts as if it were in two places at a time. This is expressed by saying that *John* is the subject of *like* at one level of representation, and then it moves as the subject of *seem* at a different level, leaving behind an empty slot which is indicated as *t*. This movement is the same that occurs in passive, as seen above in (1), or also in questions as (3).

(3) Who do you like t ?

Application of movement alone will overgenerate, producing many incorrect sentences. *e.g. Bill seems t to like t* , where *Bill* is moved from the object of *like* to the subject of *seems*, or *The children loves John* with the interpretation of (1)b. Thus, other constraints must be imposed. For example, compare the sentences in (4).

(4) a. It seems that John likes Bill
b.* It believes that John likes Bill
c. Mary believes that John likes Bill

These sentences show that the position occupied by the subject of *seem* is not an argument position, as it can contain the pleonastic *it*, an element that has virtually no semantic content, while the subject of *believe* does not allow this option, but it requires a semantically contentful element. One restriction on movement, then, is that the target of movement must not be an argument position. This constraint correctly rules out *Bill seems t to like t* , with the meaning of *Bill seems to like himself*, as the movement of *Bill* from object to subject would violate the constraint I just stated.

Consider now

(5) * It seems Mary to like Bill

This sentence shows that *Mary*, a lexically realized NP, cannot occur as the subject of the infinitival clause, while t is allowed, as we saw above. In fact, fully realized NPs can hardly ever occur as the subjects of infinitival clauses, while they can certainly occur as subjects of finite clauses. The correct restriction is achieved by assuming that all lexical NPs must receive a feature called Case (case is what distinguishes *he* from *him*. in English.) This constraint is called the Case Filter. Infinitival verbs cannot assign Case to their subjects, so lexical NPs cannot occur in this position.

Going back to our initial raising sentence, notice how these principles apply. The movement of *John* to the subject position of *seem*, from its lower position, is allowed because the subject position of *seem* is not an argument position, and is obligatory because the subject position of an infinitival verb does not assign any Case, and all lexical NPs must receive Case. These observations are captured by two principles: the Case Filter and the θ-Criterion. The former requires that all NPs receive Case, while the latter requires that the argument structure of a verb be saturated, and that all arguments in the sentence entertain at most one thematic relation. The interaction of movement, Case theory and θ-theory, together with the lexical properties of the verb *seem*, and the well-formedness principle of structure. called $\overline{\text{X}}$ theory, "generate" the raising construction without storing a raising rule in the grammar.

Of course, the relevance of this approach lies in the fact that the same principles that are used for the raising construction are used for many other types of structure. The process of passivization is very similar to raising, as passive morphology (in English the *-ed* ending of the verb) is said to "absorb" the ability of assigning accusative case to the object and a thematic role to the subject. Thus, while (6)a is correct , (6)b is not.

(6) a. John likes Bill
b.* John was liked Bill.

Since the grammar, as seen above, employs a movement rule, *Bill* moves to subject position, yielding (7)

(7) Bill was liked t (by John)

The process is very much like raising, but again no "passive" rule is mentioned in the grammar, and the same general principles do all the work.

Principle-based theories are also well equipped to capture generalizations across languages. For example, consider dependencies that involve more than one clause. It can be observed that linguistic dependencies involve either material sitting in two adjacent clauses or material separated by an unbounded number of intervening clauses. It is never the case that items, let's say 4 clauses apart and only those, are related by a long-distance dependency. This fact can be captured quite elegantly by using movement rules that perform a basic step and an iterative step. The only movement rule, move-α which links elements such as *Who* and t_i can perform a simple basic step from a clause to an adjacent clause. Or it can iterate, and as a result, displace linguistic material unboundedly far away from the source position by a sequence of basic steps. For instance, in *Who do you think t' that Mary saw t at the party?* *Who* is a displaced element that has been moved from the object position, after *saw*, here indicated by t, to the position indicated by t', and finally to the first position in the sentence. The well-formedness of such long-distance relations is regulated by the Subjacency Condition, which determines how big a single step can be. The existence of minimal pairs such as those in (8), shows that the "size" of a basic step is different from language to language.

(8) a.* Who does John wonder when Mary saw t_i t_j?
b. Il ragazzo che mi chiedo quando Maria abbia visto t_i t_j
"The boy that I wonder when Mary saw "

If long distance dependencies were captured simply by construction-specific rules, the regularity across languages would be lost, as different rules would be needed for English and Italian. This kind of cross-linguistic regularity is captured in current theories by stating a general principle (the Subjacency Condition) and then also stating a parameter (the size of each movement step), that can take different values for each language.

Thus, theories based on a small set of general abstract principles are considered explanatory, because they describe universal properties of language in a succinct way.

Current linguistic theories, GB in particular, propose themselves as models of human linguistic knowledge. In other words, they assume that universals of human language constitute a mental grammar (UG). This assumption is crucial for solving the problem of language acquisition. Languages are acquired by children in impoverished circumstances, with no negative feed-back and limited evidence, nonetheless they are acquired quite speedily and correctly. If UG is the innate knowledge of language of any human being, then the task of the child is reduced to acquiring the parameter settings for the language-dependent parameters. This set of principles and parameters that describes natural languages is called, as a mental state. *competence.*

Moreover, linguistic theory assumes that the language faculty is better understood if it is located at two different levels, a purely representational level that constitutes the competence of the language and an algorithmic level at which the representational knowledge is put to use, which constitutes the performance of the language. Competence is the level at which the *nature of the problem* is understood; performance is the level at which *how to solve the problem* is understood.

By making this (strong) hypothesis, linguistic theories acquire double status, as they are at the same time typological descriptions of languages and descriptions of a mental state. It is conceivable, and indeed it has been forcefully argued (Gazdar, Klein, Pullum, and Sag 1985), that this is not necessary, and that linguistic phenomena are still the object of justified scientific research even if viewed only as a formal system. This is certainly a tenable position. Simply, the goals of these two approaches are different. The requirement of psychological reality can only be imposed on a system that views grammars as (descriptions of) mental states.

The problem of parsing, then, becomes systematically ambiguous, as it is at the same time an investigation of computational and psychological issues. Viewing parsing as putting a mental grammar to use has often imposed stringent requirements on the forms of the possible grammars used, mostly requiring that the grammar proposed by the linguists be used directly. Were the study of grammar not viewed as the study of a mental system, there would be no need to have a precise and motivated mapping between the grammar and the parser. As long as the parser recovers the same structural descriptions that are assigned by the grammar to the strings in the language, no other stricter relation would be needed. The parser is subject to time/space constraints that are irrelevant to the grammar, and these are the only pressuring constraints to shape the architecture of the parser and the algorithms.

Thus, although it is not a necessity that grammar and parser be in a relation that captures the nature of the grammar as a mental state, this appears to be the stronger position, because it makes the minimal number of assumptions about the natural language acquisition system and the processing system. which both use the same level of representation, the competence grammar.[2]

Moreover, by making this assumption the study of natural language processing can pursue engineering and cognitive goals at the same time, since the human processor is a very efficient parser, and hence a good, and possibly enlightening model. On the other hand, engineering endeavours to solve the parsing problem as efficiently as possible will cast light on mechanisms of information processing that are relevant for the study of cognition in the computational paradigm.

In this chapter, I develop the argumentation about the relation between the grammar and the parser. I conclude that a direct correspondence is untenable and I propose a different organization. In particular:

- The tension between explanatoriness and efficiency for implementations of modular theories is illustrated and discussed. I argue that explanatoriness of linguistic theory pulls towards

[2] These issues have been debated extensively. I will not repeat the debate here in detail. See (Chomsky 1957; Chomsky 1965; Chomsky 1980b; Chomsky 1986b; Marr 1982) for arguments about the competence-performance distinction in linguistics and vision respectively, and see (Berwick and Weinberg 1983; Berwick and Weinberg 1984; Abney 1987; Van de Koot 1990) and references therein for the application of these notions to processing issues.

highly modular parsers, which use very general principles and long deductive chains of computations, but that these designs are inefficient.

- I review experimental results that point towards partial precompilation of principles as a solution of this tension.
- Observations about the structure of principles of linguistic theory and information contained in linguistic primitives provide substantiation to the idea of partial precompilation.
- A specific formulation of partial precompilation is put forth, called the Information Content Modularity Hypothesis (ICMH). This hypothesis is taken as the constraining principle on the design of the parser. It is tested with an implementation, which is illustrated in the chapters that follow.

1.2 ON GRAMMAR PARSER RELATIONS

The debate on how to draw on linguistic theories to design parsers is long-dated (Berwick and Weinberg 1984; Abney 1987; Van de Koot 1990). I review it here, to illustrate the possible relations between grammars and parsers. I follow the terminology in the cited works: *correspondence* is the relation between the function computed by the runtime grammar (the performance level) and the competence grammar. *Equivalence* is the relation between the two grammars directly.[3]

I/O Correspondence Input/Output correspondence means that the parser computes exactly the function specified by the grammar. Namely, given the same set of input strings, it outputs the same grammatical judgments, *i.e.* it fails or succeeds on the same sentences. This implies that all grammatical sentences are parsable and all ungrammatical sentences are unparsable. This

[3] I have also tried to eliminate what I consider confusing terminology. Both Abney (1987) and Van de Koot (1990) use the terms "covering" and "compiling" interchangeably. I think they denote different relations between the competence level and the performance level. In particular, covering is a (possible) relation between the competence grammar (the grammar off-line) and the performance grammar. Compilation does not imply covering at all. A grammar could be compiled (*e.g.* into a table) without any previous covering transformation. The two processes cannot be considered equivalent as covering is a manipulation of a grammar that produces a new grammar, while compilation does not. Compilation preserves strong equivalence of the structural derivations, while covering does not. The two terms can be used interchangeably only if they are used in the very loose sense of "some process applied to the grammar off-line".

theory is too strong. For example, I/O correspondence breaks down in *garden path* sentences, which are grammatical but very difficult to parse, such as (9), or easily parsable sentences that are not fully grammatical, such as those that violate the grammatical principle of Subjacency, like (10).

(9) The horse raced past the barn fell

(10) This is a plan that I do not know when they will implement.

Because of these discrepancies, a relaxed I/O correspondence is more appropriate to characterize the relation under scrutiny.

Degenerate I/O Correspondence The function computed by the parser can differ from the function computed by the grammar in two ways: it can assign a different structure to the sentence, or the competence grammar can assign a structural description to some sentence to which the parser cannot assign any structure because of limitations of the hardware, such as memory limitations. For instance, the sentence in (11) is almost incomprehensible, arguably because of memory limitations. (Miller and Chomsky 1963; Church 1980).

(11) The cheese the mouse the cat chased ate stinks.

One might also attempt to define the relation between the grammar specified by the linguist, the competence grammar, and some functional modification of this grammar which is more apt for computational purposes. Relations between grammars are called *equivalences.*

Covering Equivalence

Covering is a relation between two grammars, which maintains weak equivalence.

> "Informally, one grammar G1 covers G2 if (1) both generate the same language L(G1)=L(G2) *i.e.* the grammars are weakly equivalent and (2) we can find the parses or structural

> descriptions that G2 assigns to sentences by parsing the sentences using G1 and then applying a "simple" or easily computed mapping to the resulting output. (...) That is, if the parse of a sentence (...) is a string of numbers corresponding to the rules that were applied (...) and some canonical derivation sequence, then the translation mapping that carries this string of numbers to a new string corresponding to another parse must be a homomorphism under concatenation." (Berwick and Weinberg 1984:79)

For instance, consider a covering algorithm to transform a left-recursive grammar into an equivalent non-left-recursive one. Given a grammar G with productions P={S →Aa; A →Ab; A →a } I can transform it into a grammar G′ with productions P′={ S →aA′; A′ →bA′; A′ →a }. We say in this case that G *covers* G′. This is then an example of two grammars that are in a covering relation, one of which could be used by the linguistic representation and the other could be the grammar actually used by the parser. Covering grammars might be needed because of a particular parsing algorithm. For example, Earley's algorithm (Earley 1970) is guaranteed to work only on non-left-recursive grammars.

Strong Equivalence Another possible relation between the competence grammar and performance grammar is Strong Equivalence. This means that the two grammars must be isomorphic. This is a stronger relation than covering, because it prohibits modification of the grammar. In other words, the grammar used in the parser must be the same as the grammar specified in the theory of competence.

Given the possible relations between grammars and parsers reviewed above, which one is the *most desirable relation*? Of course, this question makes sense only if the goals of the enterprise are defined. Usually, parsing is studied from two points of view: cognitive and engineering. As I have mentioned above, work on the relation between the parser and the grammar is mostly interesting for cognitive reasons. Researchers that are interested in parsing from a purely engineering point of view do not find the need to be *faithful* to a particular linguistic theory, if this is not practically feasible. In principle, however, the cognitive and the engineering aspect can cast light on each other because both address issues of efficiency and typological validity.

Parsers that can use grammars directly are more likely to have wide coverage, and to be valid for many languages, and they also constitute the most economical model of the human ability to put knowledge of language to use. Therefore, the postulation of a direct correspondence between the parser and theories of grammar is, methodologically, the strongest position. Historically, this was the starting point, but even a very brief look at the several answers to this question that have appeared in the literature, shows that the restrictions on the required relation which in the first proposals were rather tight have been progressively relaxed.

Token Transparency In the early days of generative grammar, it was proposed that there should be a kind of rather strict *token-transparency* relation between the two levels (Miller and Chomsky 1963). This meant that if a derivation required the application of three transformations, then parsing the output of such a derivation was supposed to be three times as difficult as parsing the output of a derivation that required only one transformation. This implies that the parser mirrors exactly the derivational steps that are postulated in the competence grammar. This approach, labelled the Derivational Theory of Complexity (Fodor, Bever, and Garrett 1974, 319), was apparently contradicted by Slobin (1966)'s results. If the parser mimics exactly the computational steps of the grammar, then a passive sentence, which is generated by a greater number of transformations than an active sentence, should be more difficult to process than the corresponding active. In fact, Slobin (1966) showed, in a picture verification task, that non-reversible passives were easier (took shorter time) to process than the corresponding active sentence. Although the experiment was contradicted (Forster and Olbrei 1973), it led to a more cautious attitude towards the Derivational Theory of Complexity.[4] Weaker relations between the parser and the grammar were explored.

Type Transparency Bresnan (1978) attempts to build a model of grammar that is also better suited to be a model of language use. She requires a less stringent relation between the grammar and the parser than token transparency, what Berwick and Weinberg (1984) call *type-transparency*. She considers parsing

[4]Non-reversible passives are those sentences where selectional constraints determine the logical functions, *e.g.* subject and object, such as *The flowers are watered by the girl.* An example of reversible passive is *The girl being watched by the dog.* The fact that selectional restrictions are relevant in parsing was interpreted as supporting the *interactive* view of sentence processing. Specifically, that semantic information could influence sentence processing. On the contrary, Forster and Olbrei (1973) show that the *constancy* hypothesis is supported by experimental evidence. They show that syntactic processing time tends to hold *constant* for sentences of similar syntactic structure, even if they change in meaning. In their own words: "These facts are interpreted as indicating that the recovery of the underlying structure of a sentence is controlled by purely syntactic properties of the input." (*op.cit.*,p.319).

theory to be the realisation of competence theory if it is capable of making the necessary distinctions among types of parsing operations.

> "A realistic grammar must be not only psychologically real (...), but also realizable. That is we should be able to define for it explicit realization mappings to psychological models of language use. These realizations should map distinct grammatical rules and units into processing operations and informational units in such a way that different rule types of the grammar are associated with different processing functions. (...) Clearly, these are strong conditions to impose on a linguistic grammar." (Bresnan 1978:3).

Given her model of language, for instance, function-dependent rules and structure-dependent rules would map onto different types of realisation.

Covering Relations

Berwick and Weinberg (1983) and Berwick and Weinberg (1984) follow this approach and expand on it. They claim that the model proposed in the theory of parsing would still be a realisation of the competence level if it models a grammar that can cover the competence grammar. Thus grammar pairs can be created other than strongly equivalent grammars in a psychologically realistic parser.[5]

The advantage of this approach lies in the fact that in this way the most natural grammar for each level could be used. For example, if one were to assume that the human parser is a top down parser, and to observe that natural languages have constructions that are naturally described as left-recursive rules, for example the English genitive, it would still be possible to maintain the left-recursive formalism at the representational level, while the parser could actually use a non-left-recursive, but covering, grammar. (See also example above, of covering equivalence.)

On the other hand, as a disadvantage of this approach, one more level of indirectness is introduced between the representation and the algorithm. The

[5] Two grammars are strongly equivalent iff for the same string in the language they generate the same phrase marker. Note that here *strongly equivalent* is used in the sense introduced by Chomsky (1965), where it defines a relation between two grammars. Nothing is said with respect to the parser.

architecture of the competence grammar is justified by questions concerning natural language acquisition. But the covering grammar is not justified by such arguments. Consider, for example, modularity. A modular system is well-designed to explain data concerning natural language acquisition. This explanation, however, justifies only the modular architecture of the theory of representation. But a covering relation between grammars does not guarantee that the same partitioning of modules will be made in the covering grammar. In other words, if there are two grammars that are in a covering relation, there must also be *covering rules*, or *covering routines*, that the speaker possesses, which are however not justified by acquisition. Therefore, if we want to maintain an approach where the parser does not use the competence grammar directly, then we need to give justifications for the covering grammar in terms other than acquisition. As in the examples above, there could be assumptions about the algorithms that justify the need for the covering relation.

Principle-based Parsing and Isomorphism

The requirement of strict isomorphism as propounded by Miller and Chomsky (1963) is the null hypothesis. This was shown to be empirically incorrect, however, for the theory of generative grammar, as it was formulated at the time. One could wonder if things have changed due to the changes in the theory of grammar. Current GB theory is very different in some fundamental aspects from earlier versions of the theory. In particular, the theory of grammar as expressed by GB is a system of abstract, parameterized principles, called also modules, that interact in complex ways, so that the setting of one of the parameters in one of the modules has far-reaching consequences in the entire system.

Parallel to the shift of linguistic theory from a uniform collection of structural descriptions and structural transformations to a system of principles and parameters, parsing theory has developed the so called *principle-based* parsing approach (Barton 1984; Berwick and Fong 1990; Berwick 1991a; Berwick 1991b; Crocker 1995; Stabler 1992). This approach assumes that the principles are stated as axioms in the theory of grammar and they must be used as *axioms* by the parser. This means that the information that must be recovered by the parser to assign a structure to the input sentence is stored in a set of very general principles, and the actual structure of each individual sentence is recovered by "wading" through these principles and applying them to the input. On the one hand, principle-based parsing is explanatory and directly related to a theory of grammar. On the other hand, principle-based processing in its strictest sense prohibits grammar compilation and the use of grammar theorems, so in this sense it goes back to requiring strict isomorphism between the grammar and the parser.

1.2.1 Principle-based Parsing and Precompilation

If principle-based parsing, which is naturally implemented as a deduction algorithm, and which is deemed a highly explanatory model of parsing, turned out to be efficient, then clearly a good answer to the debate would have been found. However, this is not what has been found empirically. I illustrate the results of models without precompilation, and models with precompilation.

On-line Computation is Inefficient

Several researchers notice that principle-based parsers that allow no grammar precompilation are inefficient.

Firstly, it has been noted that, unless particular programming techniques are adopted, the problem of computing a multi-levelled theory without any precompilation, might not even terminate (Johnson 1989; Van de Koot 1991; Stabler 1990).

Secondly, experimental results show that a totally deductive approach is inefficient. Kashket (1991) discusses a principle-based parser, where no grammar precompilation is performed, and which parses English and Warlpiri by using a parameterized theory of grammar. The parsing algorithm is a generate-and-test, backtracking regime. Kashket (1991) reports, for instance, that a 5-word sentence in Warlpiri (which can have 5! analyses, given the free word order of the language) can take up to 40 minutes to parse. He concludes that, although no mathematical analysis for the algorithm is available, the complexity appears to increase exponentially with the input size. Fong (1991, 123) reports informal profiling of a parsing algorithm, which shows that an initial version of the parser, where the phrase structure rules were expressed as a DCG, and interpreted on-line, spent 80% of total parsing time building structure. In a later version, where rules were compiled into an LR(1) table, structure-building constituted 20% of the total parsing time. This same parser includes a module for the computation of long distance dependencies, which works by generate-and-test. Fong also finds that this parsing approach is inefficient.

Dorr (1987) notices similar effects in a parser that uses an algorithm more parallel in spirit (Earley 1970). Dorr notes that a limited amount of precomputation of the principles speeds up the parse, otherwise too many incorrect alternatives are carried along before being eliminated. For example, in her design, $\overline{\mathrm{X}}$ theory and the other principles are coroutined. She finds that precompiling the principles that license empty categories with the phrase structure rules reduces considerably the number of structures which are submitted to the filtering action of the other principles, and thus speeds up the parse.

The source of inefficiency stems from the principle-based design, in the sense that each principle is formulated in such a way as to be as general as possible. This "logical" kind of abstraction of each principle from the others causes a lot of overgeneration, hence inefficiency. According to Ristad (1990, 6), however, this is not surprising. He says:

> "...as is well-known, a system consisting of computational modules is necessarily inefficient, both computationally and statistically. (Restricting the amount of information to a module results in a computational inefficiency because that module is unable to prune branches in its computational tree as early as it might otherwise be able to. It results in a statistical inefficiency because a module might need to examine all available evidence in order to determine the optimal estimate. ...)"

Too Much Precompilation is Inefficient

A solution to the inefficiency of principle-based parsing is not simply precompilation, though. Experimentation with different amounts of precompilation shows that off-line precompilation speeds up parsing only up to a certain point, and that too much precompilation slows down the parser again.

The logic of why this happens is clear. The complexity of a parsing algorithm is a composite function of the length of the input and the size of the grammar. The size of the grammar is usually a constant, but for the kind of inputs that are relevant for natural language it becomes quickly the predominant factor. As Tomita (1985) points out, input length does not cause a noticeable increase in running time up to 35/40 input tokens. For sentences of this length, grammar size becomes a relevant factor for grammars that contain more than 200 rules approximately, in his algorithm (an LR parser with parallel stacks).

Both Dorr (1987) and Tomita (1985) show experimental results that confirm that that there is a critical point beyond which the parser is slowed down by the increasing size of the grammar.

Finally, in the Generalized Phrase Structure Grammar (GPSG) formalism (Gazdar, Klein, Pullum, and Sag 1985), similar experiments have been performed, which confirm this result. Parsers for GPSG are particularly interesting, because they use a formalism which expresses many grammatical generalizations in a uniform context-free format, while in GB the same generalizations are expressed by a set of heterogeneous principles. Therefore, GPSG is, practically, more amenable to be processed by known compilation techniques. Thompson (1982) finds that expanding metarules is advantageous, rather than computing them on-line, but that instantiating the variables in the expanded rules is not. Phillips and Thompson (1986) also remark that compiling out a grammar of 29 phrase-structure rules and four metarules is equivalent to "several tens of millions of context-free rules." Phillips (1992) proposes a modification of GPSG that makes it easier to parse, by using propagation rules, but still notes that variables should not be expanded.

In conclusion, a paradox arises: a parser which mirrors a principle-based theory of grammar, such as GB theory, must fulfill apparently contradictory demands: for the parser to be explanatory it must maintain the modularity of the theory, while for the parser to be efficient, modularization must be minimized so that all potentially necessary information is available at all times. As a solution to this paradox, partial compilation of principles of linguistic theory, which would reduce inefficiency while retaining modularity, can be envisaged, and is supported by the experiments mentioned above. I attempt to put forth a precise proposal on the amount of needed precompilation. Since much of the explanatory power of GB theory resides in its modular structure, I explore the concept of modularity, to find out where the modularity becomes the source of explanatoriness and where it is the source of efficiency, if at all.

1.3 MODULARITY

In this section, I start from an intuitive notion of *module*, according to which GB theory is modular, in the sense that it is constituted by different subsystems that define separate, abstract principles of Universal Grammar (UG). I explore more precisely the properties of a system of modules with respect to explanatory power and efficiency. The goal of this section is to investigate whether the intuitive notion of ***module***, in the sense with which it is used in the linguistic literature, can be formalized in a way that corresponds to the definition of module in computer science, namely that of an *informationally encapsulated* entity. The result of this investigation is negative. In fact, the subsystems of GB theory are not encapsulated at all, but they interact strongly. The terminological distinction is substantive, as it is usually assumed that modular systems are efficient *if* the modules are encapsulated. Hence, I conclude, since GB theory has in fact the structure of a strongly connected system, the interaction of all the modules cannot be efficiently computed. The presentation draws heavily on Berwick (1982) and Berwick (1985), where an abstract theory of modularity is developed. I develop the linguistic content for the abstract principles proposed by Berwick in the next section.

1.3.1 Modularity as a measure of explanatory power

A guiding belief for the development of the generative framework is that a theory that can derive its descriptions from the interaction of a small set of general principles is more explanatory than a theory in which the descriptive adequacy is obtained by the interaction of a greater number of more particular, specific principles (Chomsky 1965). This is because the size of the former theory is smaller. Each principle of the theory is designed to capture a universal generalization. Thus, each principle can generate a set whose encoding would require a much larger number of bits than the bits needed to encode the principle itself.

The classical example is the use of natural classes of distinctive features in phonology, in order to compact several rules into one. Thus, for example,

$$\mathrm{a} \rightarrow \mathrm{ae}\ /\_\_ \left\{ \begin{array}{c} i \\ e \\ ae \end{array} \right\} \text{ can be replaced by a} \rightarrow \text{æ}/\_\_[+\text{vocalic}, +\text{front}].$$

An example taken from syntax is the Case Filter. Instead of characterizing all the environments where a lexical NP can or cannot occur, the Case Filter states a generalization, since we can paraphrase the Case Filter as saying that a lexical NP can occur only in the governing domain of a category [-N].[6]

Competing theories are then ranked according to the following *evaluation* predicate (adapted from Berwick (1982, 366)):

[6]On succinctness see also the discussion on evaluation metrics in Chomsky and Halle (1968) and Rounds (1991).

> A theory A dominates a theory B for a family of languages F iff either A is descriptively adequate and B is descriptively inadequate or A and B are both descriptively adequate and $\forall$ languages $L \in$ F, the size of the description of L according to A is smaller than the size of the description of L according to B.

A modular theory that encodes universal principles has obtained a greater degree of succinctness than a non-modular theory. Therefore, according to the evaluation predicate, we consider a modular theory more explanatory than a non modular one. For the parser to maintain the level of explanatory power of the theory, it must maintain the generalizations expressed by the theory, hence its modularity.

According to Berwick (1982, 400ff), GB theory is modular in two ways: it is a multi-levelled theory, and at each level several *independent* principles are active. The former type he calls *inter-level modularity*. The latter type he calls *intra-level modularity*: separate *independent* constraints that apply separately give rise to smaller grammars.

Intra-level Modularity As far as intra-level modularity is concerned it can be shown formally (Berwick 1982, 403ff) that the size of a cascade of distinct principles (viewed as machines) is the size of its subparts, while if these same principles are collapsed the size of the entire system grows multiplicatively.

Berwick argues that principles are filters that operate on a given language L. We can think of a filter as a list of incorrect strings that must be ruled out. Such a filter is a regular set and it can be described by a finite state automaton. Its complement is also a regular set. Then the language accepted at the output of such a filter P, is the intersection of L and the complement of P. Let the size of the machine that corresponds to P, M_P, be l. Analogously, let the size of the machine related to principle Q, M_Q, be n. And so on for all the principles belonging to the theory. The total size of such a family of independent principles applying to the language L is the sum of its subparts, $O(l + m + n)$, for example.

If P and Q above are independent, then the size of the machine required to accept the intersect language is given by the cross product of the two principles. If the two principles are viewed as finite automata, then the size of the machine that represents the compiled principle will have as many states as the cross product of the states of the two initial machines. For example, if there is a principle of the grammar that states the format of the rules in the grammar, such as $\overline{\mathrm{X}}$ theory, and another principle that lists the categories for a given language, such as C, I, N, V etc., then the precomputation of these two principles will be their cross product, namely every rule format can be applied to every category.

The worst case of this compilation arises when the two principles are totally independent, while the best case arises when one of the principles can be entirely derived from the other. In this latter instance however, there is no need to keep the derived principle in the grammar. A principle that can be entirely derived from another can be eliminated from the grammar without affecting the language that can be recognised.

In sum, the maximal gain in using modularity is obtained when the principles that are compiled are not independent, while "each additional independent principle can potentially simplify the grammar by a multiplicative factor (Berwick 1982, 406)", if not collapsed.

Inter-level Modularity GB is a theory with several levels of representations, that are connected by mapping functions. For example, D-structure and S-structure are connected by the mapping function *move-α*. Intuitively speaking, in such a theory, given levels L_1 and L_2, L_2 can be constructed in two steps, first construct L_1 and then make a second pass on L_1 and construct L_2. Results from the theory of computation show that even with restricted mapping functions such a theory generates quite powerful formalisms (Berwick 1982, 400ff). (Chomsky 1957, 36) claims that the rule of English that describes coordination in a single rule cannot be stated in a vocabulary available to a context-free grammar. Rather a grammar that looks at the *derivational* history of the sentence is required. This is a transformational grammar. Consequently, *several passes* of the grammar on the sentence are allowed, first in the set of phrase structure rules to build the phrase marker and then the set of transformational rules. Thus the theory of grammar can describe complex phenomena succinctly by using different, related levels of representation.

1.3.2 Modularity as efficiency

Intuitively, the advantage gained by modularity from the point of view of explanatory power of a theory is paid with inefficiency, if the gain in succinctness is obtained at the expense of resource complexity. In other words, the reduction in size of the program is counterbalanced by increased complexity of its compu-

$$\begin{tikzcd}
 & \text{D-structure} \arrow[d, dash, "\text{move-}\alpha"] & \\
 & \text{S-structure} \arrow[dl, dash] \arrow[dr, dash, "\text{move-}\alpha"] & \\
\text{Phonological Form} & & \text{Logical Form}
\end{tikzcd}$$

Figure 1.1
The Y model of grammar depicting levels of syntactic representation and their relations

tation. But under certain conditions, this gain is greater than the loss in ease of computation, therefore modular theories are preferable. These conditions arise when the system is built by modules that are *loosely coupled* among themselves. Loosely coupled modules have a very limited exchange of information, they are *informationally encapsulated*.

Information encapsulation in itself does not guarantee efficiency yet. A modular system, whose modules are informationally encapsulated is efficient only if the *union* of the results of the computations of the modules constitutes the final result of the computation. If instead, the final result were the *intersection* of the computations, it is clear that a lot of useless intermediate computation has been performed. This latter case, however, is precisely what happens in the computation of a sentence, and it is particularly evident if we look at the inter-level modularity of GB theory, for instance.

GB theory can be considered modular only at a very high level of abstraction. Each module "abstracts away" from the computations performed by the other modules, and finding the structural description of an input string implies finding a solution that satisfies all the principles simultaneously. As has been noticed by Johnson (1989) among others, this process, observed from a procedural point of view, might not even terminate. Consider what goes on in computing the structure of a correct sentence. GB theory is constituted by several subtheories which interact with each other. These subtheories are hypothesized to operate at different levels of representation: D-structure, where elements occupy their grammatical function position (subject, object); S-structure, where elements are in different positions from the level of D-structure; the level of phonological form, PF; and the level of logical form (LF), where operator-variable and quantification are interpreted. This organization of the theory is usually represented by the "Y model", shown in Figure 1.1.

Move-α is a very general movement rule, which accounts for the mapping between levels. Consider the following example.

(12) Who does Mary like?

The sentence in (12) is a question about the identity of some human whom Mary likes. It could be answered by *Mary likes John* or shortly *John*. To capture the fact that *who* refers to the object of the action of liking, two levels of representation are postulated, and "connected" by a movement rule. Thus the D-structure representation and S-structure representation of (12) are (13)a,b respectively.

(13) a. Mary loves who.
b. Who_i does Mary love t_i ?

In (13)b *who* has moved, leaving behind a "gap", called *trace*, which receives the thematic role and syntactic feature of an object. Moreover, the fact that *who* is an operator, in the sense that it binds the range of interpretation of the trace, is represented at a different level, the level of logical form.

Thus parsing a sentence by using the modules of the theory amounts to recovering a 4-tuple (DS, SS, PF, LF) that satisfies all the principles at the four different levels of representation of the theory. A hypothetical parse relation could be expressed as (14), where the argument indicated as PF is going to be instantiated by the input string.[7]

(14) parse(DS,SS,LF,PF) $\Leftrightarrow$ d-structure(DS) $\wedge$
move-α(DS,SS), $\wedge$
s-structure(SS) $\wedge$
move-α(SS,LF) $\wedge$
yield(SS,PF).

A simple-minded generate and test procedure would guess a possible D-structure and see if it satisfies all the other constraints. This process might take a very

[7]The label *parse* might be a misnomer, as this relation is assumed to be reversible, in that the theory abstracts away from differences between parsing and generation.

long time for a correct sentence, and it might not even terminate for incorrect input, as one more application of move-α could always be postulated, in the attempt to satisfy the relation in (14). This occurs because a principle that generates structure, such as move-α has been applied without at the same time applying the principles that constrain the generation of structure, such as the Empty Category Principle, which regulates the insertion of traces in the phrase marker. Several proposals have been advanced to solve this problem. Johnson (1989) suggests using logic programming techniques, such as *partial evaluation* and *freezing*. They consist in evaluating first those conjuncts whose variables are instantiated. For example, in (14), *yield(SS,PF)* would be evaluated first: PF would be instantiated by the input string. In this way, all the solutions of the possible D-structures would be bound by the length of the input string, thus the process would at least terminate. Stabler (1990) proposes precomputing lemmas from the axiomatized theory and using them to restrict the parser. Van de Koot (1991) points out that undecidability (*i.e.* the fact that the parser might never reach the conclusion that a sentence is ungrammatical) could be avoided, if the input string were used to determine an upper bound on the number of phrases in the sentence, and hence on the number of possible D-structures and S-structures that need to be considered before failing. This proposal is based on the observation that each verb licenses a limited number of arguments (subject, object etc.), plus some structure, called functional projections (words like *that* in *I think that*), therefore the number of nodes in a tree is a (linear) function of the number of verbs.[8] Thus taking into account an unlimited number of phrase markers is unnecessary.

All the proposed solutions share one characteristic: they interleave the computation of the different levels of representation. Therefore, they do not mirror transparently the inter-level modularity of the theory. Maintaining the modularity of the theory leads to inefficiency, because the level at which GB theory can be considered modular is not the level of computation. In fact, at this level, GB is quite the opposite, it is a *strongly connected* system.

[8]The linearity of the function has been noted by Correa (1991), and it descends from a property of linguistically relevant trees, captured by $\overline{\text{X}}$ theory. Namely, $\overline{\text{X}}$ theory can only license a little subtree of at most 3 nodes for each lexical token.

1.4 PARTIAL COMPILATION BASED ON INFORMATION CONTENT

The survey of experimental and theoretical results of the previous sections leads one to conclude that the apparently conflicting requirements of efficiency and linguistic perspicuity can only be simoultaneously met by a parsing design where some linguistic information is compiled for faster access, while some other is computed on-line.

Two avenues have been pursued by others, so far, to build efficient GB parsers. In one case, a "covering grammr" is compiled, which overgenerates, and is then filtered by constraints. The compilation is done in such a way that the overgeneration is well-behaved. For instance, the correct distribution of empty categories is calculated off-line (Dorr 1993). In the other case, all the principles are applied on line, but they apply only to a portion of the tree, and are therefore restricted to a local computation (Frank 1992).

I propose to combine these two approaches by compiling the grammar, at least partially, off-line. Differently from Dorr, where the amount of compilation is heuristic and based on practical experimentation, I attempt to find a principled way of doing this. My approach shares Frank's intuition that linguistic principles have a form, which can be exploited in designing the parser. The design of the parser that I propose is based on some novel observations on the structure of linguistic principles.

Observation 1

The first observation is that, in a principle-based linguistic theory, the interaction of some of the principles expands the working space of the parser, while the interaction of other principles restricts the working space of the parser. For example, the interaction between $\overline{X}$ theory and categorial information increases the number of phrase structure rules that the parser must consider in recovering phrase structure, while the interaction between $\overline{X}$ theory, categorial information and restrictions on cooccurrence of categories in phrase structure rules (c-selection) reduces the number of phrase structure rules. This is exemplified schematically in Figure 1.2. (See also (Fong 1991; Berwick 1991a) where principles are partitioned into generators and filters.)

The first conclusion is, then, that in designing an efficient modular parser, the interaction between multiplying principles should be kept to a minimum, while the interaction between restrictive principles should be maximized. Moreover,

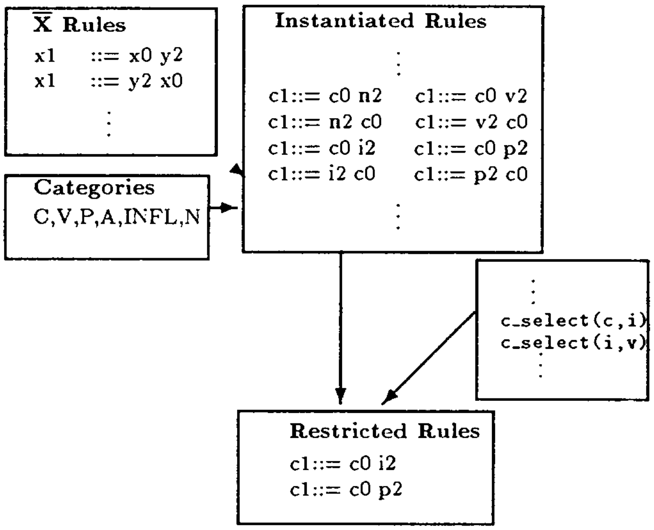


Figure 1.2
Interactions between Principles: Some of the principles interact to generate structure while some others act as filters

no generating principle should be applied without also applying the corresponding filtering principle. For example, the insertion of traces should not apply if the Empty Category Principle (ECP, see below) is not also computed. This is not a truism, as a truly logic approach would treat all the principles in the same way. (See *e.g.* Frank (1990), or the discussion of parsing as deduction in Johnson (1989).) Treating principles all on a par is unnecessary as it overlooks important computational information. The dependency of modules on each other and the best way of computing them, in what order, whether on-line/off-line, are issues that are not explicitly encoded in a grammar. However, they can be deduced in part from the "form" of the theory.

Observation 2 Second, I observe that each principle can be decomposed into separate factors. Following a suggestion by Rizzi (1990, 24), one can observe that principles of GB theory tend to have the structure shown in (15), in which the ECP is used as an example.

(15) The Empty Category Principle
An empty category x is licensed if the 3 following conditions are satisfied:

1. x is in the domain of a head H
2. the category of H $\in$ { A,Agr,N,P,T,V }
3. there is no barrier or head H$'$ that intervenes between H and x

According to Rizzi (1990, 24), the ECP must satisfy conjunctively a configuration condition, namely a condition on the shape of the tree; a substantive condition, namely a condition on the labelling of the nodes in the tree; and a locality condition, namely a condition on the subtree available for the computation of the principle. This is a single filter in the theory of grammar, but in the parser it can be computed in *pieces*, so to speak. This occurs because these conditions do not influence each other, *e.g.* the configuration is not going to depend on the categorial labelling of the head node. Note that these conditions, as they can be computed separately from each other, are modular within the little subsystem of a principle. The computation of the principles consists in checking the conjunctive satisfaction of the three conditions. To recall Berwick (1982) notion of *dependent* and *independent*, these separate conditions are independent. The precompilations of these conditions would amount to computing all the possible combinations. Thus, in this case, precomputation would lead to inefficiency.

Observation 3

My third observation regards the kinds of linguistic information that occur in the principles of grammar and that can be computed independently of each other, and it attempts to give linguistic content to the notions of *dependent* and *independent*. Berwick (1982, 400ff) states, as I reviewed above, that computation of the interaction of dependent principles leads to efficiency, while precomputation of independent principles does not. He provides examples from formal languages and automata theory. In order to transfer the discussion to the linguistic level, one must define what it means to be (in)dependent in a linguistic theory. Intuitively, one would like to claim that "modules" of linguistic theory are *independent* of each other. But, as I have briefly argued above, the modules of GB are not independent of each other.

A different solution can be found if one observes that linguistic primitives fall into different classes, according to their *content*. If we look at several of the principles of the grammar that are involved in building structure and annotating the phrase marker, we notice that what has been presented above as the structure of a principle, giving the ECP as an example, is a consistent form of internal organization of linguistic principles. θ-assignment occurs in the configuration of sisterhood, it requires a θ-assigning head, and it must occur between a node and its most local assigner. The same restrictions are imposed on assignment of Case to an NP: assignment of Case occurs in the specifier-head configuration (Chomsky 1988; Chomsky 1992), given a certain lexical property of the head ([-N]), and locally, *i.e.* within the same maximal projection. The same restriction occurs again for what is called the *wh*-criterion (Rizzi 1991), which regulates *wh*-movement, where the head must have a *+wh* feature and occur within a specifier-head configuration. Categorial selection and functional selection also occur under the same restrictions, in the complement configuration, *i.e.* between a head and a maximal projection. The licensing of subjects in the phrase marker, done by predication, must occur in the specifier-head configuration. The licensing of the empty category *pro* also requires the inflectional head of the sentence to bear the feature *Strong Agr*, and it occurs in the specifier-head configuration. The assignment of the feature [±barrier] depends on L-marking, which in turn requires that the head is lexical, *i.e.* not the node INFL or a complementizer such as *that*, and that marking occurs in the complement configuration.

Each linguistic principle can be decomposed into factors, which can be classified according to their *content.* Linguistic information belongs to 5 different classes: *configurations; lexical features; syntactic features; locality conditions; referential indices.*

(16) a. Configurations: sisterhood, c-command, m-command, ±maximal projection
b. Lexical features: ±N, ±V, ±Funct, ±c-selected ±Strong Agr
c. Syntactic features: ±Case, ±θ, ±γ, ±barrier
d. Locality information: minimality, antecedent government
e. Referential information: indices, ±anaphor, ±pronominal, binding

These I will call Information Content Classes. Notice that the 3 separate conditions that were described in observation 2 fall into different IC Classes. As we have noticed, the different factors that compose each principle are independent and they belong to a different IC Class.

These observations suggest that there is a clear structure internal to the principles of the theory that can be exploited to minimize overgeneration. I suggest a parsing design that mirrors the partitioning of information of observations 2 and 3, keeping the data dependencies that were reported in observation 1. In particular, the design of the parser that I shall discuss in the following chapters is based on three main assumptions.

No off-line interaction of configurations and categorial information

I propose a parsing design where topological information is kept separate from categorial, lexical information, in order to maximize the predictive use of lexical-invariant factors, captured by $\overline{\text{X}}$ theory. The parser uses standard $\overline{\text{X}}$ theory, compiled into a table off-line. The main parse table, though, does not encode any categorial information. Category information, together with other information of the same class, such as argument structure and subcategory, can be compiled into a table of categorial cooccurrences. The two tables of $\overline{\text{X}}$ configurations and category cooccurrence are consulted on-line. I will argue extensively in chapter 3 that the proposed partitioning of information is superior to other types of grammmar encodings. because it is more compact and more general. Moreover, this way of aggregating information is supported by psycholinguistic evidence on categorial ambiguity.

The restriction on precompilation imposed here is very stringent. It is incompatible with the use of standard context-free rules, specified with category, such as VP $\rightarrow$V NP. It is also not compatible with recent proposals in the spirit of licensing grammars (Abney 1989; Frank 1992). where structural information is encoded with each lexical item in the lexicon, thus missing generalizations on configurations. The restriction on compilation proposed here is advantageous because it keeps the size of the grammar very small. This, in turn, permits the use of space intensive techniques, such as LR parsing, which are very fast at run time.

Off-line compilation of packets of syntactic features for chain formation

The postulation and binding of empty categories involve two different stages at parsing: first the empty category must be postulated by the structure building component, and then it must be licensed by the appropriate lexical head, which assigns features to the category. I will assume that the structural licensing of empty categories in the phrase marker is done by an augmentation in the $\overline{X}$ rules, which can be computed locally to the elementary configurations defined by the $\overline{X}$ rule; the licensing by the head is done by consulting the appropriate lexical co-occurrence table. Features are assigned all together, and they interact with the structure building component to select the right feature assigment. This leads to an algorithm for chain formation that can detect the links of a chain locally, and which, compared to algorithms that do not use syntactic features (Fong 1991) is more efficient, without requiring the precompilation of the licit positions for empty categories in the phrase marker (Dorr 1993).

Interleaving of principles respects the separation between phrase structure principles and feature annotation

A problem that must be solved by a modular theory is how to partition the information in modules, which I have discussed above, and also how to interleave the modules on-line. Given the form of linguistic principles discussed above, where factors that manipulate tree geometry are separate from lexical information and from locality restrictions, the minimal hypothesis is that this same organization is kept in the parser. For example, if certain agreement annotation, like the one between the subject and the verb, is triggered only in the specifier configuration, then the on-line annotation and feature checks for agreement, will not be performed in complement configurations. There are two consequences to this assumption: the first is that feature assignment is interleaved in the parser based on configurations; the second is that locality restrictions are checked independently. In the proposed parser each linguistic class gives rise to a separate locality domain, and the domains do not interact. I will discuss this feature of the parser in chapter 5.

The design of the parser is based on a uniform set of assumptions, as all the design criteria seen so far fall under a general restriction on what linguistic information can be compiled on-line. I will call this restriction the IC Modularity Hypothesis (ICMH). In particular, if we look at observation 3, we can hypothesize that compilation of features is advantageous if they belong to the same IC Class, because they do not co-occurr freely, while it is not advantageous across IC classes as IC Classes are defined as the "factors" of a principle, thus they describe component of the linguistic description that are more loosely related to each other. Thus, we want to keep structural information separate from categorial information, as well as calculating locality restrictions separately for each type of linguistic entity (assumptions 1 and 3). On the other hand, we want features that fall in the same class to be used in packets, because only a subset of the possible co-occurrence of features do actually occur (assumption 2).

I summarise this idea, as a guideline, in the hypothesis below, and I will therefore refer to the assumption on which the design of the parser is based as the ICMH.

IC Modularity Hypothesis (ICMH)
Precompilation within IC Classes improves efficiency.
Precompilation across IC Classes does not.

In the rest of book, I discuss the advantages of storing $\overline{X}$ information separately from lexical information (chapter 3). I show that the computational advantages are a consequence of the type of grammar being used, and not a result of the particular compilation method, by showing that the results hold across compilation methods. I argue, moreover, that this particular partitioning of information mirrors some results on the time-course of the interaction of structural and lexical information in experimental psycholinguistics.

I then turn to the computation of long distance dependencies. I illustrate several algorithms that are necessary to resolve the correct postulation and binding of empty categories: I show that a particular use of syntactic feature information speeds up the parse, and I discuss the plausibility of using algorithms that require strict left to right annotation of the nodes (chapter 4). In fact, I notice that the algorithm I propose appears to be interestingly correlated to a gap in the typology of natural languages. I expand then on incrementality in parsing, and I discuss several techniques to perform the feature annotation incrementally.

Finally, I discuss locality restrictions, noticing that different local domains are related to each moved item. This observation is the most natural, given a design hypothesis such as the ICMH, because structural information, needed to postulate a trace, is independent of locality information. This idea is implemented by establishing a different local domain for each moved element, in this particular implementation the local domain are a "family" of stacks.

The parser is tested on a range of simple, complex and multiple movement cases, exemplified in Figure 1.3. This subset of constructions has been chosen because it constitutes the crucial test set for principle-based parsers, since it involves complex interactions of principles over large portions of the tree.

	TYPE	EXAMPLE
1	Simple Transitive	*john loves mary*
2	Simple Intransitive	*john runs*
3	Simple Passive	*mary was loved*
4	Simple Raising	*mary seems to like john*
5	Embedded Transitive	*john thinks that mary loves bill*
6	Embedded Intransitive	*john thinks that mary runs*
7	Embedded Raising	*mary thinks that john seems to like bill*
8	Simple Question	*who does john love ?*
9	Embedded Question	*who do you think that john likes ?*
10	Embedded Question and Raising	*who did you think that john seemed to like ?*
11	Embedded Wh-Question	* *who did you wonder why mary liked ?*

Figure 1.3
Types of Sentences

2

OVERVIEW OF THE PARSER

2.1 INTRODUCTION

This chapter presents an overview of the parser, and provides an example of how a sentence is run. With at least some high-level knowledge of the current implementation, I then turn to compare this parser to other principle-based architectures. The explanation of the technical parsing terms can be found in the Appendix, section A.3. An overall picture of the organization of the parser is shown in Figure 2.1.

Input/Output The input to the algorithm is a sentence, represented as an unannotated sequence of tokens, followed by an end-of-sentence punctuation mark. For instance, `Who did you love?`. The output consists of two objects:

1. a fully annotated binary tree, which is the parse tree of the sentence, in list format. Each node of the tree contains an identification number, the lexical features, and the syntactic features of the node.
2. a list of two lists: the list of $\overline{\mathrm{A}}$ chains and the list of A chains. Each chain is a 3-tuple *(list-of-id-numbers, lexical-features, syntactic-features)* where the list of numbers is the list of the nodes in the tree that belong to the chain, each of them identified by its number; the lexical features are the features of the head of the chain, and the syntactic features are the features assigned to the chain, namely Case and θ-assignment. Thus, the algorithm recovers the entire history of movement. By extracting the last element in the list of identification numbers that belong to each

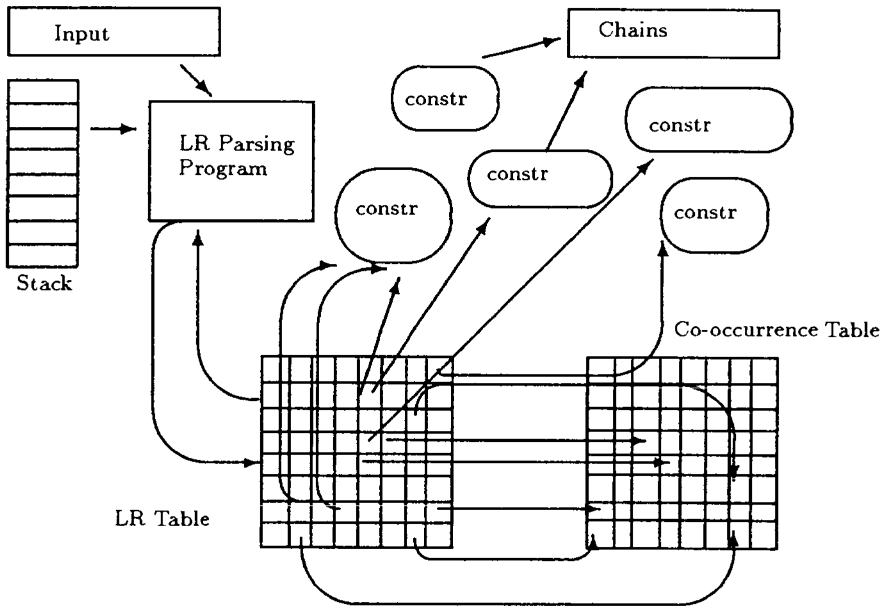


Figure 2.1
Organization of the Parser: The data structures (tables, stack and chains) are represented as rectangles. Operations on feature annotation are performed by constraints, represented as ovals.

chain, the D-structure position of the head of the chain can be returned.

Lexicon and Morphological Analyser The main components of the parser are a lexicon, a morphological analyser, and a syntactic parser. The input stream is passed to the parser token by token. For each token the parser calls the morphological analyzer, which segments words into roots and affixes according to the morphological rules of the language. Roots and affixes are stored in a lexicon, which consists of a set of records relating various types of linguistic information. Each record is a triple *(Phon, Synt, Morph)*, where *Phon* is the spelling, *Synt* is a set of syntactic features, *Morph* is a set of morphological features. The spelling field of these records, the lexeme, is used to construct a trie –or letter tree– (Knuth 1973) which allows fast access to the records. Figure 2.2 gives examples of the type of syntactic features employed.

LEXICAL FEATURES	
LABEL	RANGE
Case:	{nom, acc, dat, gen}
Category:	{n,v,adj,prep,adv,det,infl,comp}
Gender:	{m,f}
Number:	{sing,plur}
Person:	{1,2,3}
Role:	{ag,th,goal,loc,dir,ben,prop}
Tense:	{pres,past}

Figure 2.2
Features contained in the syntactic field of each lexical item

For each category (noun, verb, adjective, etc.) a relation scheme is defined which specifies feature values. A general relation scheme for the lexical categories Noun, Verb, Adjective, and Adverb is given in (17).

(17) word(Lexeme,
synt(Category, θ-Grid, Type),
morph(Form, Inflections, Morphological-Features))

After the subparts of a word have been retrieved from the lexicon, their features are merged by unification and passed on to the syntactic parser. The unification process works both as an instantiating and a filtering device. If the subcomponents of the word can unify because their features match, then many of the fields that constitute the word's lexical entry will be instantiated, while if the features do not match then the word is probably incorrect and the morphological analyzer fails. When the morphemes are correctly identified, the word is passed to the syntax. Before entering the syntax, each token is *projected* to its projection node. This means that from then on only the syntactically relevant properties of the word will be used, such as category or subcategorization frame.

X''	$\rightarrow$ Y'' X'	specification
X''	$\rightarrow$ X' Y''	
X'	$\rightarrow$ X Y''	complementation
X'	$\rightarrow$ Y'' X	
X'	$\rightarrow$ Y'' X'	modification
X'	$\rightarrow$ X' Y''	
X''	$\rightarrow$ Y'' X''	adjunction
X''	$\rightarrow$ X'' Y''	
X	$\rightarrow$ empty	empty heads
X''	$\rightarrow$ empty	empty Xmaxn

Figure 2.3
Category-neutral grammar used to build the context free backbone of the parse

2.1.1 The Syntactic Analyzer

I have argued in the previous chapter, that both linguistic perspicuity and computational efficiency, summarized in the ICMH, lead to the assumption that structural information should be computed separately from lexical-categorial information. This assumption corresponds to the division of labour between context-free methods to compute phrase structure and constraints satisfaction methods to annotate the trees.

The Grammar The full context-free grammar is shown in Figure 2.3. The $\overline{\text{X}}$ templates, which express the basic information contained in $\overline{\text{X}}$ theory, are augmented by ϵ rules. The crucial feature of this grammar is that nonterminals specify only the $\overline{\text{X}}$ projection level, and not the category.

The Parse Cycle The LR algorithm is encoded in a *parse* predicate, which establishes a relation between two sets of 5-tuples, as shown in (18).

(18) $t_i \times s_i \times a_i \times c_i \times pt_i \rightarrow t_j \times s_j \times a_j \times c_j \times pt_j$

In (18) $t_k \in T$, the set of input tokens, $s_k \in S$, set of states in the LR table, $a_k \in A$ the set of attributes associated with each state in the table, $c_k \in C$ the set of chains, *i.e.* displaced element, and $pt_k \in PT$ the set of tokens predicted by the co-occurrence table. Three stacks are used, to shift elements that have been reduced: a stack for the states traversed so far; a stack for the attributes associated to each of the nodes; a tree stack of partially recovered trees.

Tables

The grammar shown above is compiled into an LALR(1) parse table (Aho and Ullman 1972) . The LR(k) parsers developed for programming languages are deterministic. This means that in each state of the parser there is a unique next state, i.e. the LR table has no conflicts. Modifications are necessary to treat natural languages with this method. Namely, the LR(k) parser table can have more than one action for each entry. The parser can handle arbitrary context-free grammars, but is no longer deterministic.[1]

Co-occurrence Table In order to reduce the amount of nondeterminism, some predictive power has been introduced. The information that belongs to the IC class *lexical features* has been compiled into a co-occurence prediction table.

By looking at the current token, at its category label, and at its subcategorization frame, the number of choices of possible next states can be restricted. For example, if the current token is a modal verb,(I_0), the next token to be reduced must contain a V, as the only possible complement of I_0 is VP. Or if the current token is a verb, and the LR table allows the parser either to project one level up to V', or it requires the creation an empty object NP, then, on consulting the subcategorization information, the parser can eliminate the second option as incorrect, if the verb is intransitive. The choice of possible next states is restricted to one, in most cases, by comparing the multiple options compiled in the LR table, which are deduced exclusively from the $\overline{X}$ grammar, to the possible next token, which is encoded in a co-occurrence table.

2.1.2 Constraints

As a result of using a category-neutral context-free backbone to parse, most of the feature inheritance and feature annotation that could be encoded in the nonterminals is performed in this system by conditions on attribute annotation associated with each context-free rule, the *constraints*.

[1]Several methods have been researched to augment LR(k) parsers, so that they can deal with natural languages, mainly based on the observation that NLs are close to LR (Tomita 1985). In some cases a graph of parallel stacks is used (Tomita 1985; Tomita 1987). Fong (1991) adopts a backtracking approach.

This is in fact a necessary modification to render the LR method more suitable for natural language parsing, as it keeps the phrase structure rule set minimal. Most of the work in parsing consists in constraint checking, rather than manipulating phrase structure rules. Rather, the phrase structure rules constitute a *context-free backbone* which serves to anchor a set of grammar constraints, as shown in (19).[2]

(19) $X \rightarrow Y\ Z \Leftrightarrow f(X,Y,Z)$

The lefthand side is an expression of $\overline{X}$ theory. The formula f on the righthand side represents grammatical constraints on the features of the nonterminals which must be satisfied to license the production. Each rule is associated to a subset of the universal constraints that form the grammar, as shown in Figure 2.4. The set of constraints must be satisfied for the rule to apply.

Figure 2.5 shows what features are manipulated by each condition, while Figure 2.6 shows the pool of available constraints. The parser built this way is compact, because the number of context-free rules needed to parse a language is small. The separation of $\overline{X}$ rules and constraints results in a considerable reduction of grammar size, as can be observed. If all the categories and all the $\overline{X}$ structures had been combined, the grammar size could have been equal to the cross products of the rules and the constraints, in the worst case. Moreover, since only highly abstract rules are used, language-independence is achieved.

[2]This division of labour between phrase structure and constraint checking is also found in many *unification-based* grammars such as PATR-II (Shieber 1986), which employ only a minimal context free *skeleton*, augmented by unification equations.

RULE	CONSTRAINT
sentence	θ-criterion case filter
specifier	categorial selection predication percolation of features
complement	categorial selection θ-marking case marking unification with chain percolation of features
modifier	categorial selection percolation of features
adjunct	categorial selection percolation of features
unary xmax	unification with chain percolation of features
unary head	assignment of features to the specifier percolation of features complement head selection
empty head	categorial selection (ECP) feature percolation
empty xmax	licensing (ECP) locality condition

Figure 2.4
Interleaving of $\overline{\text{X}}$ Rules and Other Principles

Constraint	Features
θ-criterion	θ-role
case filter	case
node labelling	{ AI, $\overline{\text{A}}$I, AH, $\overline{\text{A}}$H, AFoot, $\overline{\text{A}}$Foot }
chain select	$\overline{\text{A}}$ A
chain unify	Case. θ-role
head feature percolation	category { N,V,C,I,...}
feature absorption	case, θ-role, passive
θ-marked	θ-role, $\pm$referential
case-marked	$\pm$case
c-select	category { N,V,C,I,...}
locality	$\pm$barrier
licensed empty xmax	$\pm\gamma$

Figure 2.5
Filtering Principles and their Range of Features

2.2 AN EXAMPLE

I give an example of a parse, in order to illustrate the main characteristics of the algorithm. Examples of other constructions that are handled by the parser are given in Appendix B. The LR algorithm is traced step by step by giving the state name, the next input token, the contents of the stack, and the chain list. Feature assignment and chain formation are illustrated by showing incremental feature assignment, the node labelling procedure, and the postulation of empty categories.

Here below I show the actual output of the algorithm. For the sake of brevity, only the most relevant states are shown in full and commented.

```
INPUT TOKENS: who does john seem to like ?
state:    5
token:    w0
stack:    (who, 6, m, s (n,[],[bar (2), wh]), fts (_, _))
Abar chains:
          (who, 0, m, s (n,[],[bar (2), wh]), fts (_, _))
A chains:
```

CONSTRAINT	FUNCTION
θ-criterion	checks if all chains in the chain list have received a θ-role
Case filter	checks if all chains in the chain list have Case
node labelling	determines what kind of chain link the current node is: head, intermediate, foot
chain selection	selects chain to unify with current node
chain unification	unifies node with selected chain
head feature percolation	consults cooccurrence table and determines cooccurrence restrictions among heads
θ-marked	marks node with available θ-role
case marked	marks node with available Case
c-select	categorial selection
is-a barrier	checks if maximal projection is a barrier
license empty head	checks features of closest lexical head
licensing head	finds a lexical head to license a maximal projection
locality	checks that the maximal projections between antecedent and empty category are not barriers

Figure 2.6
The Constraints that are used to decorate the tree

The token *who* is recognized as a maximal projection. The format `(who, 6, m, s (n,[],[bar (2), wh]), fts (\_, \_))` indicates the node whose lexical content is `who`, the identification number is 6, which is a maximal projection `m`, the syntactic field shows the node is a noun with an empty θ-grid, directly projected to a bar 2 level, and which is a *wh* word. The features related to its status as a link in a chain are uninstantiated. However, the node is also recognized as the member of an $\bar{A}$ chain. The chain has not been assigned any features yet.

```
state :  9
token :  w0
stack:   (john,9,m,s(n,[],[bar(2),proper]),fts(_,_))
         (does,8,h,s(comp,infl,[]),fts(_,_))
         (who,6,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
         (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
         (john,2,m,s(n,[],[bar(2),proper]),fts(_,_))
```

The two words *does, John* are shifted onto the stack. *John* is also recognized as the potential head of an A chain because of its configuration (SpecIP).

```
state :  1
token :  w0
stack:   (seem,3,h,s(v,[(_,nil,th),(_,ext,nil)],[rais]),fts(_,_))
         (_,10,h,s(infl,[],[empty]),fts(_,_))
         (john,9,m,s(n,[],[bar(2),proper]),fts(_,_))
         (does,8,h,s(comp,infl,[]),fts(_,_))
         (who,6,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
         (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
         (john,2,m,s(n,[],[bar(2),proper]),fts(_,_))
```

An empty *INFL* is pushed onto the stack. In contrast to many other parsers (Macias 1991; Johnson 1989; Frank 1992) this parser does not operate on annotated input, or input which is previously decomposed into morphemes. The fact that a lexically empty INFL node is part of the input must be computed on-line. (The same is true for empty, base-generated complementizers.) The word *seem* is also pushed onto the stack. Its θ-grid encodes the fact that raising verbs do not assign Case to the object, nor a θ-role to the subject. Subcategorization properties other than the θ-grid are not encoded here, but are predicted on-line.

```
state :  6
token :  w0
stack:   (seem,12,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
         (_,10,h,s(infl,[],[empty]),fts(_,_))
         (john,9,m,s(n,[],[bar(2),proper]),fts(nil,ext))
         (does,8,h,s(comp,infl,[]),fts(_,_))
         (who,6,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
         (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
         (john,2,m,s(n,[],[bar(2),proper]),fts(nil,ext))
```

External case to the word *John* is assigned structurally, in virtue of the fact that the verb is tensed, without waiting for a reduction to apply. (See Chapter 4 for a discussion of this issue.) The θ-grid slot is marked as expended `(d,ext,nil)`: this is needed in order to compute one half of the θ-criterion, which checks that all roles have been assigned. Marking the expended θ-roles is crucial from a computational point of view, as it guarantees that each node will license at most a small number of other nodes, some of which possibly empty. From a computational point of view, a principle like the ECP, which ensures that empty categories are licensed only in the environment of nonempty θ-assigning heads, guarantees termination of the parse, as only a number of empty categories which is linearly related to the number of heads in the sentence can be postulated. (I discuss licensing of empty categories in chapter 4.)

```
state :  9
token :  w0
stack:  (_,14,m,s(n,[],[empty]),fts(_,_))
        (seem,12,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,10,h,s(infl,[],[empty]),fts(_,_))
       13
        (john,9,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (does,8,h,s(comp,infl,[]),fts(_,_))
        (who,6,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
        (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
        (john,2,m,s(n,[],[bar(2),proper]),fts(nil,ext))
```

The co-occurrence prediction of *seem* states that if the next token is an infinitival marker, then an empty maximal projection is expected. The flag 13 is the sentinel pointer inserted in the stack to delimit the left context related to the empty category 10. (The fact that 13 relates to 10 is recorded in the database, where a constraint on locality is posted.) The locality restriction states that intervening maximal projections between the empty category and the sentinel cannot be barriers. (See Chapter 5.)

```
state :  6
token :  end_of_file
stack:  (like,18,h,s(v,[(_,acc,th),(d,ext,ag)],[]),fts(_,_))
        (to,16,h,s(infl,v,[inf]),fts(_,_))
        (_,14,m,s(n,[],[empty]),fts(ag,nil))
        (seem,12,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,10,h,s(infl,[],[empty]),fts(_,_))
       13
        (john,9,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (does,8,h,s(comp,infl,[]),fts(_,_))
        (who,6,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
        (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
        (john,2,m,s(n,[],[bar(2),proper]),fts(nil,ext))
```

The tokens *to, like* are shifted onto the stack. The subject of *to like*, the empty category number 14, receives the θ-role *agent*, but no case, as the subject of an infinitival is not case marked.

```
state :  9
token :  end_of_file
stack:    (_,20,m,s(n,[],[empty]),fts(_,_))
         (like,18,h,s(v,[(_,acc,th),(_,ext,ag)],[]),fts(_,_))
         (to,16,h,s(infl,v,[inf]),fts(_,_))
         (_,14,m,s(n,[],[empty]),fts(ag,nil))
         (seem,12,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
         (_,10,h,s(infl,[],[empty]),fts(_,_))
        13
         (john,9,m,s(n,[],[bar(2),proper]),fts(nil,ext))
         (does,8,h,s(comp,infl,[]),fts(_,_))
        19
         (who,6,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
         (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
         (john,2,m,s(n,[],[bar(2),proper]),fts(nil,ext))
```

Another empty category is postulated and dropped into the stack, and a different left context for locality is set up, indicated by the sentinel 19.

```
state :  4
token :  end_of_file
stack:  (_,21,p,s(v,[(d,acc,th),(_,ext,ag)],[]),fts(_,_))
        (to,16,h,s(infl,v,[inf]),fts(_,_))
        (_,14,m,s(n,[],[empty]),fts(ag,nil))
        (seem,12,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,10,h,s(infl,[],[empty]),fts(_,_))
       13
        (john,9,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (does,8,h,s(comp,infl,[]),fts(_,_))
       19
        (who,6,m,s(n,[],[bar(2),wh]),fts(nil,nil))
Abar chains:
        (who,[0|20],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
        (john,2,m,s(n,[],[bar(2),proper]),fts(nil,ext))
```

Reductions begin. θ-role and Case are assigned to the object of *like*, which is recognized, since it has Case, as the foot of an $\overline{\text{A}}$ chain. The set of $\overline{\text{A}}$ chains is searched for a chain to unify. ([0|20] are the indices of the nodes participating in the chain.) The unification of features generates a completely annotated $\overline{\text{A}}$ chain.

```
state :  8
token :  end_of_file
stack:  (_,25,m,s(infl,v,[inf]),fts(_,_))
        (seem,12,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,10,h,s(infl,[],[empty]),fts(_,_))
       13
        (john,9,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (does,8,h,s(comp,infl,[]),fts(_,_))
        (who,6,m,s(n,[],[bar(2),wh]),fts(nil,nil))
Abar chains:
 (who,[0|20],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
 (john,[2|14],_,s(n,[],[bar(2),proper]),fts(ag,ext))
```

The subject of the infinitival is identified as the foot of an A chain, because of its features, and unified with the head of the chain (shown by the indices [2|14]).

```
state :  10
token :  end_of_file
stack:     (_,35,m,s(comp,infl,[]),fts(_,_))
Abar chains:
  (who,[0|20],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
 (john,[2|14],_,s(n,[],[bar(2),proper]),fts(ag,ext))
```

After a series of reductions the well-formedness conditions on chains are checked. All chains are well-formed, as they all have a θ-role and Case. The sentence is accepted with the following tree structure.

```
Parse Tree :                        comp2
                                      |
            .-------------------------.
            |                         |
           n2                       comp1
            |                         |
            .     .-----------------------.
            |     |                       |
           n2   comp0                   infl2
            |     |                       |
            .     .      .-------------------.
            |     |      |                   |
           who  comp0   n2                 infl1
                  |      |                   |
                  .      .     .----------------.
                  |      |     |                |
                does    n2   infl0             v1
                         |     |                |
                         .     .      .------------.
                         |     |      |            |
                       john    e     v0          infl2
                                      |            |
                                      .    .---------.
                                      |    |         |
                                     v0   n2       infl1
                                      |    |         |
                                      .    .    .--------.
                                      |    |    |        |
                                    seem   e  infl0     v1
                                                |        |
                                                .      .----.
                                                |      |    |
                                              infl0   v0   n2
                                                |      |    |
                                                .      .    .
                                                |      |    |
                                               to     v0    e
                                                       |
                                                       .
                                                       |
                                                       like
Abar chains:
         (who,[0|20],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
         (john,[2|14],_,s(n,[],[bar(2),proper]),fts(ag,ext))
```

2.3 RELATED WORK

In this section I review some related work, to set the proposed parser in context. The number of so-called GB parsers is suspiciously small, mainly due to the rather fluid state of the formalization of the theory, which renders faithful implementations very difficult. A few researchers have built GB parsers, and have grounded their proposals rather firmly in linguistic theory. I discuss some of them here.

2.3.1 Abney 1986, Abney 1989

In Abney (1986), the idea is proposed that syntactic parsing is driven lexically, based on licensing relations. A *licensing parser* uses the properties of lexical items to recover the structural relations of the elements in the sentence. Each item in the sentence must be licensed to be grammatical. Licensing relations are triples, *(Direction, Category, Type)*, attached to the projection of each lexical item. *Direction* is the directionality of the licensing relation, *Category* is the category of the licensed item, and *Type* is the type of the licensing relation: it can range over *Subjecthood, Functional Selection, θ-assigment, and Modification.*[3]

The parsing algorithm to scan a sentence is simple.

(20) Proceeding from left to right, examining two words at a time, assign to the words whatever relation can be assigned.

This parsing procedure, supplemented with some heuristics, exhibits the same preference that human subjects exhibit in parsing locally ambiguous sentences, such as *While she was mending the sock fell off her lap.*[4].

This algorithm is too simplified, as all arguments of a verb are going to be represented as sisters to the verb and there is no mechanism to represent long distance dependencies. Abney (1989) proposes an augmentation by LR states. The goal of parsing with Licensing Grammars (LS) is to postulate nodes in

[3]They will be abbreviated to S, F, Theta, and M in the examples below.

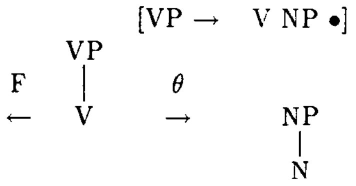
[4]For instance, in the sentence *While Mary was mending the sock fell off her lap*, the NP *the sock* is usually read as the object of the verb *mend*, rather that the subject of *fell* (Frazier and Rayner 1982; Kimball 1973)

a parse tree in an order which is more plausible psycholinguistically than LL or LR grammars.[5] Since a simple LS parser does not have enough global knowledge about the grammar, and it would never recover if it made a wrong choice on ambiguous input, it is augmented by two mechanisms: LR states, attached to each node, and a set of heuristics that enable the parser to recover from incorrect analyses.

This proposal is unsatisfactory in two ways: firstly, taken on its own grounds, this parser fails to fulfil the stated goals of representational parsimony, and it makes some incorrect psychological predictions. Secondly, it is based on unnecessarily negative assumptions about other, more efficient, parsing methods. We illustrate them in turn.

Abney does not discuss how the LR states, to be attached to each node, are to be constructed, but there are only two choices: either they are projected from the lexicon, or they are compiled from a global grammar, which the parser can look up. Clearly, if the LR states were projected from the lexicon, they would be useless. The LR states are needed to keep track of global information, to compute if the shifting of the next token could eventually result in a legal reduction. But this kind of computation can be performed only if the input token can be compared against some independently stored grammar. Thus, it must be that Abney means the LR states to be compiled from some grammar which encodes global information about phrase structure, and which is used to guide the parse.

If this is true, the LR states reduplicate some of the information about category and structural licensing that is stored in each lexical item in the LS. For example, the representation for a verb phrase will look like the following.

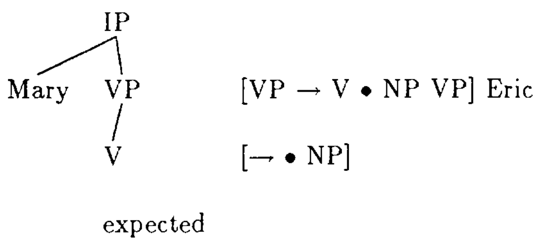


[5] Abney (1989) argues that LR parsers are implausible because they do not start building nodes until the end of the sentence is reached, in a purely right-branching structure, while psycholinguistic evidence shows that humans integrate material from the input as soon as possible (Frazier and Rayner 1982). Against such a restrictive view of incremental LR parsing, see Stabler (1991), Shieber and Johnson (1993) and also the discussion below, in this chapter and chapter 4. LL grammars introduce spurious ambiguities, by which humans do not appear to be bothered. For example, an LL grammar would find it difficult to choose between the rules VP →V NP and VP→V NP PP.

Thus, this representation is not parsimonious, and it fails to fulfill one of the initial goals for the development of LS grammars, namely to eliminate the redundancy between the lexicon and the phrase structure component (Stowell 1981). It is true however, that it maintains an order of creation of nodes different from the one in an LR parser, and especially that it does not suffer from the (supposed) lack of incrementality on right-branching structures typical of LR parsing algorithms.

The LR states are also needed to recover from failure, if the parser starts a false analysis and must backtrack. A particular heuristic is proposed by Abney, which should account for the difference between strong and weak garden paths. The heuristic scans the right edge of the built tree in search of an abandoned LR state that has a possible continuation of the next input item. The parser cannot keep an agenda of abandoned states, but it is limited to those structurally available on the right edge of the tree. Weak garden paths are those for which such a state exists, while strong garden paths are those for which it would have been necessary to keep track of abandoned states which are no longer on the right edge of the tree. This is an operation that the parser cannot perform, and strong garden paths arise.

For example, take the sentence *Mary expected Eric to leave*. At the point where the parser has seen *Eric*, it is in the following configuration.



The notation `expected [→ • NP]` means that *expected* is subcategorized for a noun phrase, thus it predicts an NP. (Abney calls these LS states.)[6]

[6] I have followed Abney's notation, thus instead of IP, we write, simplifying, NP VP. Note, however, that a more correct characterization of phrase structure rules would require to use nonterminals such as IP, and subsequently to be able to know that the first token in an IP can be an NP. Moreover, this appraoch requires translating the grammar given by linguistic theory into a covering grammar, more suitable for parsing purposes.

Attaching the NP to the main verb is preferred, thus the state `[VP → V • NP VP ]` is abandoned. When the next input word is seen no continuation is possible, thus the parser must backtrack. As there is an available abandoned state, which has a legal continuation on an input token of label V, the state is retrieved, and *Eric* is attached as the subject of the embedded clause. Abney illustrates the example in (21)b.

According to this mechanism, the following sentences should both be weak garden paths, and they should be on a par, since in both cases the attachment as complement is chosen first, thus the alternative option of closing a constituent earlier is abandoned. This in turn results in the abandoned state being available on the right edge of the rightmost tree, which is the one being built. However, the judgments appear to be different, in that (21)a does not cause any processing difficulty, while (21)b is a garden path.

(21) a. Mary expected Eric would leave
b. While she was mending the sock fell off her lap

Abney's parser does not capture correctly the psycholinguistic evidence that it was designed to handle. With respect to this latter criticism, we note that, although we do not put forth an independent theory of garden paths, the model we propose, an LR parser, is compatible with several deterministic models of sentence processing, *e.g.* the minimal committment model in Weinberg (1993). Moreover, the mechanism for incremental feature assignment discussed below in chapter 4, makes the LR architecture much more compatible with models that maximize grammatical relations any given point in the parse (Gibson 1991; Pritchett 1992).

Finally, we note that the main motivation for not adopting the central control of an LR parser, while still using many other compilation mechanisms, might not be very strong. Several arguments have been proposed against the idea that LR parsing is not incremental. I advance some ideas here and discussed them more thoroughly in chapter 4.

First, LR parsers are criticized because they build nodes in a rightmost fashion. But this is not necessary. While the rules of the grammar are traced in a rightmost derivation, the input is scanned from left to right. A parser that does not build trees in the standard fashion, namely building nodes only on rule reduction, could start asserting structural statements, for instance D-theoretical statements (Marcus, Hindle, and Fleck 1983), in an order that is more in accordance with the input.

Second, LR parsers are criticized because they do not assign features incrementally. Again, this is true only if feature annotation is done exclusively on reduction. I argue extensively in chapter 4 that this is not necessary. (Stabler (1991),Shieber and Johnson (1993) also argue that LR parsers are compatible with incremental interpretation.) Moreover, the stack of an LR parser, in virtue of tracing a rightmost derivation, encodes c-command. Thus, all the nodes below the top nodes in the stack, those that constitute the internal structure of the nodes in the stack, are no longer visible, and they can be redirected to a semantic interpreter (Berwick and Weinberg 1984).

2.3.2 Fong 1991

The main goal of Fong's work is to build a system to explore the computational properties of principle-based grammatical theories, mainly a tool for the grammar writer. This system enables the user to specify a grammar which is automatically compiled into a parser. Fong defines a set of tools, which can be partitioned in two classes for expository purposes: one set enables the user to specify the grammar in a highly abstract language, very close to the definitions used by the linguist; another set of tools automatically compiles the grammar specifications into a *family of parsers*, in a way suitable for experimentation. I describe them in turn.

The Representation of Principles

The first set of tools addresses the issue of what constitutes a *transparent* representation of the theory of grammar. Fong argues in favour of a representation which is close to the definition of the linguist, is efficiently executable, and abstracts away from choices of control strategy. The representation of linguistic principles consists of two components, in this system:

1. top-level macros provided to the user to define linguistic principles;
2. linguistically motivated primitives.

Two top-level macros are provided, to expand the definition of principles automatically: the former is used to encode quantification over tree structures, while the latter is used to encode operations, which are compositional on trees.

Universal Quantification over Configurations Consider, for instance, the case Filter, which states that all lexically realized NPs must receive Case: in a universally quantified configuration, the property of being a lexical NP and of being Case-marked must hold. The macro expands this filter into two different forms, depending on whether the Case Filter is compiled or not, (see below the discussion on *interleaving*), but these different expansions are transparent to the grammar writer, who can just ignore implementation issues.

Compositional Operations on Trees This macro is used in compositional definitions. Certain parser operations establish non-local structural relations. For instance, the principle of free coindexation, states that every NP can be coindexed with any other NP anywhere in the sentence. The non-local nature of this principle poses two problems, which are solved automatically and transparently by this macro: they pose the problem of how to compute such unbounded relations in a *sensible* way, namely a way that emits *all* indices, but no duplicate indexing; it also poses the problem of how to compute such unbounded relations incrementally. If free indexation cannot be computed until the entire tree structure of the input sentence is recovered, this means that the principle does not influence the structure building part of the parse at all. The compositional macro provides a solution to the latter problem by computing a compositional principle inductively over tree structure, *i.e.* independently over each tree substructure.

The Recovery of Phrase Structure

The second set of tools supports different kinds of automatic compilation of the principles specified by the grammar writer into a *family of parsers*, in order to study issues related to control and efficiency.

Fong argues that the mechanism to recover phrase structure must be efficient because all structural relations are based on phrase structure; because there could be more than one phrase structure per sentence; and also because optimization of control (principle ordering or interleaving, see below) depends on the optimality of the structure recovery process, namely how fast the parser can detect an error in the input. For these reasons, Fong adopts an efficient architecture, an LR(1)parser. LR(1) parsers are deterministic, and they fail as soon as it is possible to fail on incorrect input. The LR(1) parse table however, requires modifications, as natural languages are not LR languages.[7] Fong introduces two main modifications: first, each entry in the action table can hold an indefinite number of actions; second, every clausal node must cover a string which contains at least one non-null terminal symbol (in order to guarantee termination).

The parser uses a finite state automaton (FSA) and three stacks as data structures. The FSA consists of two tables: a transition table and an action table. The three stacks record the states of the machine, partially constructed constituents, and contextual information.

Principles of the grammar that filter incorrect structure can be applied to independently recovered phrase structure skeleta or they can be interleaved with the phrase structure rules, and be applied on reduction of each rule. In the former case, all the principles are applied at the end of the process which recovers phrase structure, thus they require ordering. Fong observes that some issues related to principle ordering arise (p. 162).

(22) a. What effect has principle ordering on parsing a sentence?
b. Are some orderings better than others?
c. Is it possible to predict which the best orderings are?

As an answer to (22)a, one can note that principles of grammars are divided into *filters* and *generators*. Principle ordering can make a relevant difference, if the ordering is such that overgeneration is minimized. In other words, if an ordering is chosen that applies filtering principles immediately after the corresponding generating principles, then a speed up will occur. As an answer to the second question, Fong observes that there cannot be any *global optimal ordering*. An optimal ordering is one that eliminates ill-formed structure as soon

[7] Tomita 1985 has shown that natural languages are close to LR. Although natural languages are ambiguous, therefore they are not LR, they do not exhibit many features of CFLs that would distance them from LR even more than ambiguity: for instance, natural languages do not exhibit *dense ambiguity* or *infinite recursion*. Tomita argues that natural languages can be parsed by LR parsers, and that LR parsing is the most efficient for practical purposes. See also Fong (1991) for LR parsing and Pereira and Wright (1991) for fast algorithms for practical purposes.

as possible, thus applies a given principle first. But different sentences violate different principles, hence there cannot be a global solution to optimizing the ordering of the principles. Finally, it is possible to predict the optimal ordering for each sentence by using computationally cheap cues, which infer from the input which principle is most likely to fail. Different orderings are not going to have any effect on the parsing time of a correct sentence, because a correct sentence must pass all the filtering principles, but they are going to make order-of-magnitude differences in parsing incorrect input. The experiments with this dynamic control strategy are very interesting because they constitute a bench mark for comparisons for other control strategies. In particular, the principle ordering problem arises only if an (almost) generate-and-test procedure is adopted, which is supposed to be very inefficient. Hence, other control strategies can be compared to it, to see that predicted speed-ups really occur. Our proposal cannot be compared to dynamic control, but rather it is static, and therefore more similar to the other strategy that Fong explores.

A different control strategy interleaves the principles with the rules that build the structure, so that filtering constraints are applied as early as possible. The main question that arises with regards to principle interleaving is how to interleave the principles in a way which is transparent to the theory of grammar.

Fong argues against a naive model, (*i.e.* a model that applies every principle every time some structure is built), because many principles are irrelevant (*i.e.* vacuously true), but they still require computation to determine that the premises do not hold. The classes of structures that never satisfy the preconditions of a given principle can be determined by using information about the range of possible phrases to which this principle may apply. This range of phrases is called a *type*. Fong defines the type of a principle as being the set of category labels that are associated with a configuration. Once the type is computed, each principle is interleaved off-line with the appropriate rules, depending on the category labels on the left side of the rule, by an automatic interleaver. The task of the interleaver is to recast principle definitions into a series of specialized predicates for each category in the type of the principle.

The experimental results of the performance of the interleaved parser show that parsing time increases dramatically as more principles are interleaved; the increase is greater when *generators*, *e.g.* Move-α, are interleaved; adding filters, such as the case Filter, has no relevant effect. This result is surprising, as one would expect that, by interleaving, the computational burden of the parser should be reduced. However, the computation added by the interleaved principles turns out to be costly, because a lot of work is done to check constraints on analyses which, in the end, will be discarded.

It is interesting to note that Fong's results on faithfulness and transparency to the grammar are mainly negative. In his experiments with different compilations of the grammar, he explores dynamic ordering and principle interleaving with rather surprising results. The former results in significant speed-ups if the right ordering is found, but it is based on totally heuristic cues, which are external to the grammar, while the latter, which is based on the grammatical concept of category to interleave principles, does not speed up the parse, not even when filtering principles are applied, Of course, the first avenue to explore would be to reformulate the definition of type. In particular, it seems more natural and more useful to interleave principles based on configurations rather than category. For example, Fong notices that θ-assignment can occur only in complement configurations of verbs and adjectives, and specifier of NP. Thus, he concludes, the type is given by the union of all the prime and maximal projections of all these categories. The filtering principles will then be called by a "hook" indexed into the category, for instance `reduceNP`. The experimental results show that using principles to prune the search space is not effective in this kind of interleaving.

Our partitioning of IC Classes leads us to a different interleaving model, in two main respects. First, the "hooks" of the principles into the rules are the configurations, *i.e.* filtering principles are interleaved according to $\overline{X}$ theory, and according to the co-occurrence table. Hence the search space is restricted both by category and by configuration. Secondly, Fong applies all the filtering principles independently, one at a time, without precomputing the interaction of some of the filtering principles off-line. The result of this design choice is that interleaving some principles can cause major slow-downs. Consider, for instance, Trace theory: as soon as this principle is interleaved, a major slow down occurs because many false trace positions are postulated. This is, however, an artifact of the way Trace theory is implemented, as functional determination, therefore the incorrect prediction of a trace cannot be detected locally. Our partitioning of principles into IC Classes, however, points in the direction of structural computation of empty categories, which can be determined locally. For example, Case information can be used immediately to postulate a *wh*-trace and to start the search for a possible antecedent. If no antecedent is found the structure can be immediately discarded.

Fong's approach to the relation between the parser and the grammar, and to what constitutes a principle-based parser is somewhat different from standard, but well-supported by a sophisticated implementation. The usual interpretation of the label *principle-based parser* is that of a parser which encodes the theory of grammar transparently, and uses the grammar on-line.(Barton 1984; Abney 1986; Johnson 1989; Frank 1992; Kashket 1991; Berwick and Fong 1990).

In this work, however, Fong takes the stand that a principle-based parser can operate at two (representational) levels. One level consists of the specifications of the linguist's grammar, this level is the direct, declarative encoding of the grammar. To this purpose, tools for the expression of universal quantification and compositional principles are provided to the grammar writer. The second level is the compiled version of the grammar that the linguist writes, namely the parser, in which many techniques of parsing and compiling optimization are used. Fong, therefore, does not appear to share the belief that a principle-based approach to parsing forbids grammar compilation. Rather, Fong adopts the idea that grammar compilation can be used, as long as it is completely automatic and transparent to the user. Such point of view makes sense practically. It enables the linguist to be a user of such system, without being concerned with implementation issues. Moreover, it provides a tool to test the theory in a more formal way. This kind of approach does not address to claims of *psychological reality* that have been made for principle-based parsers (Abney 1989; Johnson 1989).

Although this work shows convincingly that GB can be used as the grammatical representation for an efficient parser, it often relies on extralinguistic augmentations. First, the optimal ordering of linguistic principles is based on extra-linguistic cues, as we have already observed. Clearly, one could wonder how such cues are learned (by a machine or by humans), and how they are evaluated, since it is crucial for them to be computationally cheap.

Another mechanism which is not very plausible, cognitively and linguistically, is the use of unbounded look-ahead. The trace licensing mechanism in the LR table consists in the insertion of a dummy nonterminal (see below, chapter 4), which triggers unbounded lookahead before inserting a trace in the phrase marker. The result of such a powerful device is that garden paths, which could be modelled, at least in part, are predicted to cause absolutely no difficulty. Another case in which Fong's parser is more powerful, arguably, than the human parser is in processing cyclic movement. Although it has been extensively argued that an LR parser might exhibit the right architectural features to account for the existence of cyclic movement (Marcus 1980; Berwick and Weinberg 1984; Berwick and Weinberg 1985), Fong's parser has been augmented in such a way that it can postulate unboundedly distant dependencies. In fact, the stack mechanism has been augmented by an *environment stack* which enables unboundedly distant tokens of the input to communicate. This stack is equivalent to a HOLD cell in an ATN (Woods 1970; Wanner and Maratsos 1978).

Thus, a fully automatic and user-transparent compilation process is not sufficient to be "faithful" to the representation, unless we ignore completely the supposed cognitive relevance of the competence level. In other words, Fong's compilation is faithful to the letter, but it fails to capture the content of the organization of the competence grammar. It might be also be the case that a transparent compilation is not completely beneficial. Some of Fong's criticism against manual compilation is unconvincing, because it forces him to implement a parser which is unnecessarily inefficient, for example, in the instance of structural computation of empty categories vs. functional determination. Fong's determination of empty categories is a generate-and-test approach, because he rejects the insight in Correa (1988) that empty categories can be efficiently determined if computed on the basis of structural properties. We present in chapter 4 evidence in favour of using syntactic features to compute chains. For example, the foot of an $\bar{A}$ chain can be readily identified by looking at Case information. Moreover, psycholinguistic experiments show that the human processor makes use of syntactic featural information to make structural decisions as soon as possible (De Vincenzi 1991).

Finally, Fong does not address the issue of implementing the LR parser, in a way that supports incremental interpretation. This lack of incrementality is certainly a problem for psychological plausibility, but also for practical applications. In favour of this approach, however, we can point out that Fong has developed tools that make it possible to use GB, and the large body of linguistic literature set in this framework, for many practical purposes.

2.3.3 Dorr 1990, 1993

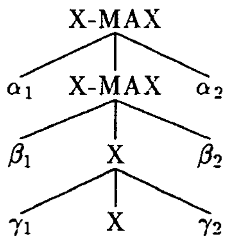
Dorr (1990),Dorr (1993) presents UNITRAN, a bidirectional translation system for English, Spanish and German, whose syntactic component is based on GB theory. Translation is done by mapping the syntactic structures created by the parser onto a universal level of lexical conceptual structure. This same level provides the input level for generation of the target language. The main feature of this translation system is its high degree of compositionality and parameterization. Syntactic differences across languages are captured by the interaction between principles modelled after GB, which are common to all languages, and parameters, that can acquire distinct values for different languages. Not only is this representation valid cross-linguistically, but it can also be used for generation, with small changes, as a result of the compositionality of the approach, since syntactic and lexical conceptual information are computed separately. I concentrate on the syntactic parser, which is also described in Dorr (1987).

The parser produces a syntactic structure for the input sentence of the source language, which is then passed to a component for the analysis of its lexical-conceptual structure. The lexical-conceptual structure of the target language is then determined. The target language sentence is produced from this by means of the target language lexical selection and syntactic realization routines.

There are two main stages to parsing a sentence: all the phrase structures compatible with the input are recovered first, and then constraints are applied.

Phrase Structure

The first stage makes use of a phrase structure grammar, which is compiled off-line by computing the interaction of several modules of GB theory and of the language-dependent parameters that have been selected for the particular source language in question. The relevant modules are $\overline{X}$ theory, Trace theory, and the language parameters that set constituent order, the choice of category, and the availability of clitics in the language. These pieces of information are then compiled into phrase structure rules which have the format shown below, adapted from Dorr (1990, 39).



In the phrase-structure template, α_1 and α_2 stand for positions for adjunction of maximal projections, β_1 and β_2 are positions for arguments, specifiers and complements, and γ_1 and γ_2 are positions for head adjuncts, such as clitics. As can be noticed, this kind of phrase structure rule is more informative than the typical $\overline{X}$ template. Dorr (1990, 40) lists several reasons for augmenting the phrase structure component this way: first, the co-routining mechanism that was necessary in a previous version (Dorr 1987) is greatly simplified; second, this schema is reversible, as much of the information that is needed for parsing or generation is encoded in the phrase structure component; third, it is faster than less informative rules, because by encoding some of the constraints in the phrase structure rules, the number of incorrect structures that are going to be postulated on-line is greatly reduced.

The grammar so produced is then consulted by an augmented Earley's algorithm (De Marcken 1990). Earley's algorithm is a tabular parsing method that consults the grammar directly. Since many parses are pursued in parallel, at the end of this stage, more than one phrase structure could be compatible with the grammar and the input.

Constraints

The second stage of the parse consists of the application of the other modules of the theory: Movement theory, Case theory, Trace theory, Binding theory, and θ theory.

After the trees compatible with the input have been recovered by the Earley algorithm, the distance between the landing site of a moved element and its source position is checked, in accordance with Subjacency. Case theory checks that all nominal phrases receive abstract Case, which means that they are the complement of a verb or a preposition, or they are the subjects of a sentence. Trace theory is parameterized, so that languages like Spanish or Italian, which allow the subject of a sentence to be understood, can still be parsed by the same mechanism which would mark such usage as incorrect in German or English. Finally, Binding theory construes the reference of nominal and pronominal elements.

θ theory is the module of GB that controls the correct distribution of arguments of verbs. In Dorr's system it is the interface with the semantic processor. The interpretation of a sentence, in this system, is performed by reconstructing the lexical-conceptual structure (Jackendoff 1983; Jackendoff 1990), which is then used as input to the generation of the sentence in the target language. The generation process runs through the same modules in reverse. After the lexical-conceptual structure in the target language has been selected, syntactic phrases are generated, according to their syntactic canonical realization in the language, and they are attached to the $\overline{\text{X}}$ template seen above, and finally all the feature annotation is checked with the same modules that impose constraints in parsing. Parameters related to constituent order and adjunction procedures are relevant to this process.

This system encodes GB principles in an indirect way, in that it compiles them into complex phrase-structure rules off-line. The relation to the theory is maintained because the same syntactic structures are produced that the theory predicts, the algorithms are modularized in a way that strongly resembles the modules of the theory and parameters are used to capture differences between languages. However, there is no principled definition of precompilation, and parameter specifications and settings do not always reflect linguistic theory directly.

The two-stage architecture of Dorr's system addresses the issue of *principle interleaving* by expanding all the possible structures compatible with the rules, and then applying all the non-local constraints to the forest of trees thus generated. This approach is sound and complete and it can be practical if the algorithm to build phrase structure is very fast. Moreover, Fong shows experimentally that in principle-based parsers of comparable complexity, this approach, where the non-local constraints are checked last, is more efficient than a truly *interleaved* one. However, this solution has some shortcomings. First, it does not provide principled reasons to precompute only certain pieces of information and not others. The solution to this problem here is heuristic. Second, it fails to provide interesting suggestions on how to limit the search space, when building phrase structure.

The main difference between Dorr's approach and the approach presented here is that Dorr uses extensively covering grammars off-line. In particular, the positions where an empty category could occur are precomputed into phrase structure rules, and so is categorial information. This gives rise to a rather large grammar. Since Earley's algorithm is used, an even larger number of parsing states is generated at run time. The space of possible parsing states is then pruned down by applying filtering principles. On the other hand, the design proposed here attempts to reduce the space of parsing states much earlier in the parsing process.

On the other hand, there are also similarities. The definition of IC Classes and of the ICMH is an attempt to provide a principled explanation for Dorr's experimental result (Dorr 1987) that only partial precompilation of principles gives rise to speed ups.

Dorr's model is particularly interesting, as it is the only principle-based system dealing with cross-linguistic variation by using the parameters envisaged by the theory in an efficient design.[8]

[8]Kashket (1991) is a parser for English and Warlpiri, which is implemented as a totally nondeterministic theorem prover. As such, it is designed with different goals and it does not address the same questions as the present work.

2.3.4 Frank 1992

Frank (1992) puts forth an interesting proposal for a GB parser expressed with the formalism of Tree Adjoining Grammars (TAGs) (Joshi 1985).

Frank observes that all principles in the theory of grammar undergo some locality restrictions. Therefore, he proposes that locality restrictions are not to be expressed explicitly, but they are part of the *metagrammar* and therefore should be captured by the formalism that is adopted. In particular, he proposes that these locality restrictions can be captured in the TAG formalism naturally. All the principles of the grammar should be statable over elementary trees as defined in TAG. An elementary tree (ET) is, according to Frank, a lexical element, with its arguments and its functional projections. Thus, for example, a verb with its arguments and the Inflection and complementizer projection is an elementary tree. A DP with its argument is another kind of elementary tree.

This proposal is supported with evidence from linguistic theory, by showing that phenomena that require explicit locality constraints in GB can still be captured in this framework; with evidence from acquisition, by showing that stages of language development can be seen as the development of the manipulation of the elementary trees provided by TAG; and with evidence from parsing, by showing that TAG provides a formal definition of *bounded search* and thus GB theory recast in these terms can be efficiently parsed. I concentrate mainly on the proposal related to parsing.

Frank claims that his parser is psychologically plausible because it builds syntactic structure and semantic interpretation incrementally; moreover, the parser is efficient because the representation it uses (TAG) guarantees a bounded working space.

The design of Frank's parser rests on the observation that GB principles are of two kinds: those that express relations between adjacent nodes in a graph, and those that express constraints on the well-formedness of structures that span more than two adjacent nodes in a graph. The former is captured by licensing relations, like Abney's, while the latter by TAG. Structure is built from left to right while scanning the input, as a result of satisfaction of constraints that are directly attached to each node. When a node is projected from the input, it is paired with two sets of licensing features: a set of *gives* and a set of *needs*. *Gives* are those feature bundles that can license other nodes in the tree, while *needs* are feature bundles that must be satisfied for the projected node to be licensed and thus incorporated into the structure that is being built. The association of a node to a set of *gives* is determined lexically while the association with a set of *needs* is done by principles of the grammar, depending on the category of the node. For instance, a node whose label is V is lexically associated with a set of *gives* that constitute its thematic grid, whose cardinality determines how many arguments the verb takes and whose content determines what arguments it takes. An NP, on the other hand, is associated with a set of *needs* that require for the node to receive Case in order to be licensed, as stated by the Case filter, and to receive a θ-role, as stated by the θ Criterion. The Extended Projection Principle for instance is encoded as a *give* of type *Subjecthood* of an I node.

In this way, phrase structure is built without context-free rules, by simply requiring that all *gives* and *needs* on a node be satisfied when a node is no longer on the right frontier of the structure that is being built. Those nodes whose *gives* and *needs* are not satisfied at this point are considered to be part of a chain, and are pushed onto a *trace stack*. This method for building phrase structure requires that all empty categories must be explicitly licensed, therefore it disallows intermediate traces. Intermediate traces are a by-product of the assumption in GB that long movement must be *successive cyclic*. Such an assumption guarantees that what appears to be unbounded long movement is in fact a sequence of shorter steps, each of which obeys the standard locality constraints on derivations, namely Subjacency. In this framework no need arises for the Subjacency constraint on movement, and consequently no intermediate traces, because locality is a primitive in TAG.

Frank observes that constraints in GB are of two types: very local and bounded non-local. Determining the correct interaction between the local and non-local constraints is important for the explanatory power and the efficiency of the parser. It constitutes the problem of *principles interleaving* that we have mentioned in reviewing Fong's work. Frank proposes to perform the interleaving at the point where each piece of the structure corresponds to an elementary tree, as defined in TAG. Whenever the parser has built enough structure so that it constitutes an elementary tree, either auxiliary or initial, it detaches this piece of structure, by performing reversed licit TAG operations: adjunction and substitution. The unadjoined/unsubstituted pieces of structures then undergo well-formedness checks, and if these checks are successful, they are passed to a semantic module for interpretation. According to Frank, this approach has two very desirable consequences: it supports incremental semantic interpretation and it guarantees that the parser will always operate in a working space of bounded size.

The design of this parser solves some of the issues that we are addressing in an elegant and coherent way. However, some of the claims about it might be too strong.

First of all, we note that all the necessary augmentation to make the LS grammar work make it equivalent to a phrase structure grammar like $\overline{X}$, possibly with the disadvantage of being procedurally encoded instead of being declaratively encoded as a grammar of rewrite rules. For note what happens when a lexical item is projected from the lexicon. Each item is projected at three different levels, (which correspond to the three levels of $\overline{X}$ theory), then each bar level is coupled with the appropriate *gives* and *needs*, which corresponds to having branching possibilities only at those two levels, and moreover corresponds to binary branching, as only one satisfaction of each *give* is possible. As this same procedure is repeated for all lexical items independent of category, this corresponds to the fact that $\overline{X}$ captures a cross-categorial generalization.

One disadvantage of LS grammars, which we already reviewed above in discussing Abney, is that they do not provide enough global knowledge to recover from incorrect analyses. The possible augmentations envisaged by Abney, which Frank does not consider, are representationally redundant and do not correctly model human performance. Frank's parser as it is presented, could not parse head-final languages, as the parser has no ability to stack arguments while waiting for the head. The augmentation with a stack, is possible, but then the parser would incur the problem that lead Abney to use LR states. Namely, in order to parse head-final languages, a "shift" operation must be added to the current available operations of the parser. As there could always be a licensing head in the right context, which would license a left-branching structure, the "shift" operation is always correct. But then, the parser might reach the end of the input before realizing either that it pursued an incorrect analysis, in the case of ambiguous input, or that the input is ill-formed. Thus, this augmented parser could not recognize errors as soon as they are encountered, in violation of the requirement of incrementality that Frank states at the onset. On the other hand, the fact that the present version of the parser does not incur this problem is an artifact of it not being sufficiently general. If both the stack and the augmentation of the LR states were added, the empirical coverage could be achieved, but the architecture would be redundant and it would not fulfill one of the major desideratum of Frank's design, namely to avoid grammar compilation.

As a third point, it is not clear that the entire array of linguistic facts can be captured by this parser. It has been shown that the problem of determining licit coreference in a grammatical theory where the assignment of indices is done by free indexation, such as GB is, is NP-hard (Fong 1990). Fong suggests that binding theory can be seen as the way to limit the domain of possible coreferring items in the same sentence. This is not true for principle C of the binding theory which states that an R-expression must be free, which means not coindexed with any c-commanding elements. It does not appear likely that Frank's parser can maintain its limitation on the working space and still compute all the possible coindexations, without assuming, like many other parsers, that coreference is computed at a different level from the computation of phrase structure.

Finally, the claim that the parser can work in linear time is based on an analysis that does not take into account some computations. Several issues are at stake. First of all, the structure building component of this parser parses a sentence in $O(n^2)$ time on non-ambiguous input, while other techniques, would parse in $O(n)$ time. Second, the claim is made that polynomial time can be reduced to linear time by reducing the size of the work space to the finite bound determined by an ET defined by TAG. But this computation of the complexity does not include the actual work to determine that a given substructure is an ET. Since this checking is done for each attachment and all the nodes in the tree are checked, then this procedure will have an $O(n)$ best case, which arises when all attachments are undone as soon as they are done, thus always checking over a single-node tree. The worst case occurs when no excisions are made and the final tree corresponds to an elementary tree (this is an actual possibility in TAGs). In this case the procedure is, again, $O(n^2)$, as the introduction of ETs has not reduced the work space. More common cases will range between the two, and they will take linear time, with increasing larger constants for smaller numbers of elementary trees. Moreover, the claimed linear time of the algorithm does not take into account the needed computations to determine whether, at each point of excision, an unadjoining or an unsubstitution must be performed. Franks does not discuss the mechanism, but from his examples it is clear that cases can arise when there is ambiguity of choice between the two operations, and choosing the correct one requires knowledge about the structure which is going to be built several tokens ahead.

For example, consider the following example *Randy seemed to like the pizza yesterday* (Frank 1991, 21). Up to the I_0 projection of the embedded clause, (*i.e.* the expended input is *Randy seemed to*), the structure built by the parser is shown in Figure 2.7.

Frank points out that this structure includes two independent ETs, the two I′s. He comments:

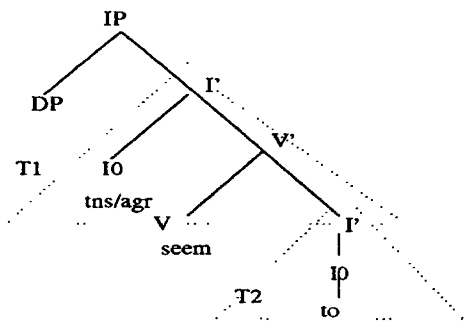


Figure 2.7
Parsing with TAGS

> "Hence, we must reduce the structure in some way. There are two possibilities for how this could be done. We might undo a substitution of the I′ node dominating *to* or else we can undo an adjoining of a structure recursive on I′. The problem with this possibility is that it would give rise to a structure in which the DP has an unsatisfied theta need, and has no further possibilities for satisfaction. In the case of the unadjoining, the DPs become part of the lower clause's elementary tree and can receive its theta role from there. (Frank 1991, 21)"

Since the parser is at a point of excision, this means that the computation about the ET T_1, the one which is being excised, depends on some features of T_2, a different ET. Thus the parse working space is not limited to a single elementary tree. Finally, it is not clear how the recognition of an elementary tree is performed. In the absence of a compilation procedure on the set of elementary trees, that could guide the parser, whenever a node is attached and the tree is checked to see if an elementary tree has been built and needs to be detached, the entire set of elementary trees must be explored, in the worst case.[9]

[9] LR parsing techniques for TAGS have been developed (Schabes and Vijay-Shanker 1990). Frank does not mention them, but their proposal would not comply with Frank's ban on grammar compilation.

2.3.5 Crocker 1995

Crocker (1992), Crocker (1995) describes a principle-based parser, founded on the tenet that the parsing architecture must maximize incremental interpretation. The parser is the implementation of the theory of grammar, as directly as possible, therefore it is modular. Modularity permits distributed concurrent processing of four modules, which are defined on the basis of the representation they use: phrase structure, chains, argument structure, and co-reference.

Psycholinguistic research has proposed models that attempt to minimize syntactic complexity of the parse, either by adopting principles of representational parsimony (Frazier and colleagues), or by assuming a very efficient algorithm (deterministic parsing.) Crocker suggests instead that the parser does not work independently of semantic interpretation, and that in fact, incremental interpretation determines the behaviour of the parser. As far as the structures that the parser builds are concerned, the main consequences is that a fully connected structure must be built at each step, and that all structure that can be built must be. This general principle also influences the time course of parsing, as the parser commits to an analysis a soon as a plausible interpretation is possible. Thus the syntactic component is not "autonomous", since it is influenced by semantics and pragmatics. However, the syntactic component is modular, because each "representational", or "informational" type requires a different, specialized vocabulary. The modules do not communicate among each other, and each of them is sensitive only to one particular type of input. (For instance, the chain building module does not have access to the category and the level of the chain element).

The Psycholinguistic Model

Each of the representational modules that comprise the parser has a specific behaviour. Two of these treat information – phrase structure building and filler-gap dependencies – that have been studied experimentally.

The Phrase Structure (PS) module builds structure based on constituency and sisterhood relations, and it serves as the basis to compute chains, coreference, and argument structure. The behaviour of this module must conform to attested psycholingustic evidence. In particular, it assumes minimal lexical information, supported by experimental results arguing that subcategorization information is not immediately used by the parser (Mitchell 1987). This module also incorporates structural attachment preferences, defined as a preference for argument attachment (AA) and a preference for base-generated $\overline{\text{A}}$ positions over moved $\overline{A}$ positions. The former strategy accounts for PP attachment preferences, reduced relative clauses, and the preference for an NP complement over a clausal complement. The latter strategy is supported by evidence in verb-final languages. In German, a PP which is ambiguous between attachment to an NP and a verb, is preferentially attached as the modifier of the noun. If a long adverb intervenes between the PP and the verb, the parser commits to this analysis. If the sentence turns out to require a complement analysis of the PP, speakers report a garden path. Crocker correctly notices that this range of data could not be explained by a model that attempts to satisfy lexical requirements at each step, such as Pritchett (1992). Both strategies are motivated by the principle of incremental interpretation, as both expand the interpretable θ-grid as much as possible. They obey the modularity design because they both operate exclusively on structural information.

The empirical evidence accounted for by the chain module is best described by Frazier's "Active Filler Strategy": once a filler has been seen, rank a gap above all other options. In order to interpret this trace-eager behaviour in a modular parser, Crocker assumes that postulation of traces is performed by the PS module, which "... continually attempts to posit traces which are sustained only if they can be incorporated into a well-formed chain structure. (p.103)" Interestingly, Crocker observes that filled-gap effects (Crain and Fodor 1985), as well as some of the attachment phenomena in head-final languages, can be explained by thinking of traces as belonging to a different "dimension" than the rest of the string. A trace must be postulated as soon as it can be, but then overt material can still intervene between the trace and the licenser. Crocker proposes a strategy that posits traces as soon as the antecedent has been seen, called the Active Trace Strategy, and connects it immediately to the tree. This strategy explains effects of VP attachment in verb-final languages, such as German and Dutch, where verb complements seem to be integrated into the structure even before finding the base position of the verb, which has moved to second position in the sentence.

The Computational Model

The properties that the parser must have, make the computational model non-trivial. In particular, it is interesting to explore solutions for the combined assumptions of modularity and incrementality, for the assumptions of information encapsulation according to representational types, and the direct use of grammatical principles.

Adopting the framework of parsing as deduction (Pereira and Warren 1981), Crocker assumes that parsing is like theorem proving, and a grammar is a set of axioms. The crucial property of the deduction system, which makes it effective, is that the principles of the grammar are defined as conditions on locally well-formed branches.

In this implementation, each linguistic representation type

> "... is a meta-interpreter, whose task is as follows
>
> 1. play the logical role of structure generators, proposing instances of uninstantiated structure for the particular representation.
> 2. sustain only those structures that are well-formed formulae with respect to the 'necessary' axioms/constraints.
> 3. determine the control strategies and preferences, where multiple structures are licensed in accordance with performance theory.
>
> (p. 118)

By using the predicate `freeze`, each word goes through the whole set of modules at a time, simulating concurrent processing and incrementality.

The PS module is a meta-interpreter for $\overline{\overline{X}}$ theory and traces, that implements, at least in part, the Active Trace Strategy. It uses a left corner algorithm to combine bottom-up instantiation of structure to top-down postulation of traces for which an antecedent has already been seen. The Chain module determines heads, tails, and links of a chain, by using the **append** relation. This module does not see the whole tree, but only that part which is visible for chain building purposes: the trace and the elements that do not occupy a base-generated position (Spec IP, Spec CP). The Principle of Incremental Interpretation requires to construct a maximal partial thematic structure as the input is received.

The parser presented here and Crocker's model are, quite clearly, very similar as far the relation between the grammar and the parser is concerned. In both proposals a modular design is adopted that does not conform *directly* to the theory, but rather the so-called *modules* of the theory are recast in computational terms. In fact, the relevant classes that are defined operate on very similar objects. However, my partitioning does not rely on the particular representation used. The spirit of the hypothesis is that linguistic theory is formed by heterogeneous types of information, and that the representation used to describe them is a derived concept.

The two parsers differ in their purpose, as Crocker's goal is simulating human behaviour. Consequently the issues addressed and the model proposed are actually quite different in the implementation.

First, Crocker adopts a parsing model that might be computationally inefficient, since he requires incremental interpretation. Admittedly, he adopts a point of view where global requirements might at time slow down particular components of the language understanding system. Since solid data on human behaviour are actually lacking, all positions are in fact justified. I have adopted the opposite position, since I assume that the correct model is a fast parsing algorithm, which works autonomously from the semantic interpreter. I find that exploring the issue of efficiency is crucial to develop models that are at the same time satisfactory engineering systems and plausible models of human performance.

Secondly, Crocker differs in the solution he proposes for the *principle interleaving* problem, which is done by modifying the flow of control of the theorem prover by using the *freeze* predicate. While this is probably a more plausible model of human behaviour, as it assumes concurrent processing of indepedent modules, it raises some computational issues with respect to empty categories. Empty categories pose a particularly complex problem to the kind of model that Crocker assumes, as it is not clear how the parser would behave on incorrect input. Assuming that the parser backtracks when it finds an error, a free postulation of empty categories might send the parser into an infinite computation of empty categories if the conditions for the their postulation are not immediately checked. Crocker does not discuss this point, but indeed his predicate **ps_ec_eval** must contain some checking of this sort for the parser to run. But notice that if this is true, then Crocker is forced to use lexical and thematic information in his PS module, decreasing the kind of encapsulation he wants to build in his parser.

This model is very interesting, however, and especially the treatment of empty categories for verb-final languages and the postulation of the Trace Active Strategy are particularly enlightening, in their separation of the postulation of traces from the linear recognition of the input.

3 THE PHRASE STRUCTURE COMPONENT

3.1 INTRODUCTION

In chapter 1, I have presented an hypothesis on how to partition the different types of computations that need to be performed to recover the syntactic structure. I have argued that linguistic theory itself, in particular the *content* of the principles, provides a guideline to perform this partitioning.

In the present chapter I substantiate this hypothesis and attempt to test it. The methodology is computational. A parser has been implemented, and the size of the data structure and the amount of nondeterminism in the tables will be used as indices of the effectiveness of the proposed phrase structure building procedure. The main result is that precompilation of structural configurations and categorial information increases the size of the grammar without reducing the ambiguity that has to be resolved on-line. I shall also discuss the literature on the interaction between syntactic and lexical ambiguity to argue that there is evidence that, in certain instances, lexical co-occurrence restrictions can be computed without taking structural information into account, which supports separating $\overline{\text{X}}$ rules from categorial information, as I suggest.

I assume that the partitioning of the information predicted by the ICMH can be implemented in an LR(k) architecture. The choice of this type of parsing architecture is independent of the ICMH, although strongly motivated by parsing factors. First, LR parsers have the valid prefix property, namely they recognize that a string is not in the language as soon as possible (other parsing methods have this property as well, for instance Schabes (1991). A parser with this property is incremental, in the sense that it does not perform unnecessary work, and it fails as soon as an error occurs. Second, the stack of

X''	$\rightarrow Y''\ X'$	specification
X''	$\rightarrow X'\ Y''$	
X'	$\rightarrow X\ Y''$	complementation
X'	$\rightarrow Y''\ X$	
X'	$\rightarrow Y''\ X'$	modification
X'	$\rightarrow X'\ Y''$	
X''	$\rightarrow Y''\ X''$	adjunction
X''	$\rightarrow X''\ Y''$	
X	$\rightarrow$ empty	empty heads
X''	$\rightarrow$ empty	empty xmaxn

Figure 3.1
Category-neutral Grammar

an LR parser encodes the notion of c-command implicitly. This is crucial for fast computation of chains. Third, LR parsers are the fastest on unambiguous input.

3.2 THE DATA STRUCTURES AND THE PARSING ALGORITHMS

3.2.1 The Grammar

In accordance with the definitions of IC Classes, notions such as headedness, directionality, sisterhood, and maximal projection are compiled and stored off-line, because these notions belong to the same IC Class, *configurations*. These features are compiled into context-free rules in our parser. The full context-free grammar is shown in Figure 3.1. These $\overline{X}$ templates express the basic information contained in $\overline{X}$ theory, namely that phrase structure rules are endocentric, that they encode the notion of maximal projection, and that they are order independent (see Appendix A for definitions.)[1] Two crucial features of this grammar are that nonterminals specify only the $\overline{X}$ projection level, but not the category, and that rules are binary branching.

[1] It is important to note that not any encoding of context-free rules will do the job. Kornai (1983) has shown that $\overline{X}$ grammars have formal properties that are different from CF

The basic $\overline{\text{X}}$ rules are augmented by rules licensed by Trace theory, in particular by that part of Trace theory that deals with configurations. According to this module of GB, only heads and maximal projections can be empty categories. Trace theory does not specify the distribution of empty categories, which is the result of the interaction of several other modules. Consequently, the possible positions of empty categories are not restricted off-line in the current parser. In the absence of other constraining principles, empty categories can occur in every position where a maximal projection or a head can occur.

Although interested in achieving succinctness of representation, I have not chosen to use an ID/LP representation (Gazdar, Klein, Pullum, and Sag 1985). In the ID/LP formalism, the righthand side of rules does not encode linear order. Shieber (1984) shows that Earley's algorithm ((Earley 1970) can be applied to these grammars by applying the dotted items to the rules in multiset notation, instead of using ordered rules. The saving in grammar size of this formalism is considerable. For instance, a single unordered rule such as $S \rightarrow a\ b\ c\ d\ e$ corresponds to $5! = 120$ rules in an ordered CFG. The number of state sets is reduced likewise. This is advantageous because not all linear orders need be spelled out in advance, since not all of them arise, but they are determined by a given input. However, the worst case running time of this algorithm is an exponential function of the number of input tokens, while Earley's algorithm worst case is proportional to the cube of the input. As Barton (1987, 80ff) shows, the worst case for UCFGs arises when the grammar is ambiguous and includes λ rules. Natural languages have these properties.

> "... Informally, the reason why Shieber's algorithm sometimes suffers from combinatorial explosion is that there are exponentially more possible ways to progress through an unordered rule expansion than an ordered one. When disambiguating information is scarce the parser must keep track of all of them." (Barton 1987, 83)

Barton shows also that the exponential time complexity is inherent in ID/LP recognition, independent of the choice of algorithm. In other words, ID/LP recognition suffers from unnecessary combinatorial explosion. (We refer the reader to Barton, Berwick, and Ristad (1987) for proofs and to Rounds (1991) for further discussion of the relevance of these results.) Since the main reason to

---

grammars, hence a parser that uses context-free rules, which do not obey the $\overline{\text{X}}$ convention explicitly, are not going to capture the exact properties of natural languages.

adopt the ID/LP formalism is grammar succinctness, I have not adopted it and I have pursued succinctness of representation by partitioning the information usually encoded in a context-free grammar along different lines, specifically by separating $\overline{\text{X}}$ information from categorial information.

3.2.2 The Parser

Separation of structural from categorial information is obtained by compiling two tables that the parser consults on-line: an LR table, where $\overline{\text{X}}$ information about the shape of the tree is stored, and a second table, where information about the cooccurrence of category and subcategory information is stored. The LR algorithm takes an unannotated sentence as input and, by consulting these two tables and other auxiliary data structures, produces an annotated tree as output.

The LR Table

The grammar is compiled into an LALR(1) parse table. The table can have more than one action for each table entry, since the grammar is not an LR grammar. The parser can handle arbitrary context-free grammars, but is no longer deterministic.

Three stacks are used: a stack for the states traversed so far; a stack for the attributes associated to each of the nodes in the state stack; a tree stack of partially recovered trees.

The Parse Cycle

The encoding of the parse cycle, described in Chapter 2 is different from the standard encoding of an LR parser, as it establishes a relation, while an LR parser is deterministic, hence it establishes a function from an input pair, a state and a token, onto a new state. Our parser is more elaborate and less restrictive, because it imposes conditions on the attributes of the states and it allows nondeterminism. Nondeterminism is reduced by using the table of lexical co-occurrences.

Table of Lexical Co-occurrence and Left Corner Prediction

The grammar used in the LR table is a pure $\overline{\text{X}}$ grammar, and it is not instantiated by category. Thus, it is underspecified with respect to the categories in the input, and many instances of LR conflicts can be simply teased apart by consulting knowledge about cooccurrence restrictions, which would be stored in the rules themselves if ordinary context-free rules were used.

For example, if the current token is an I_0, the next token to be reduced must contain a V, as the only possible complement of I_0 is VP by functional selection. Or if the current token is an intransitive verb, and the transition table allows the parser to either project one level up to V′ without branching, or it requires the creation of an empty object NP, then, on consulting the subcategorization information, the parser can eliminate the second option as incorrect. For instance, consider the standard context-free rules in (23), which are instantiated by category, which correspond to the three cases mentioned above.

(23) a. $\mathrm{V}' \rightarrow \mathrm{V\ NP}$
b. $\mathrm{V}' \rightarrow \mathrm{V}$
c. $\mathrm{I}' \rightarrow \mathrm{I_0\ VP}$

Since our parser uses only $\overline{\mathrm{X}}$ rules, the rules in (23) correspond to (24).

(24) a. $\mathrm{X}' \rightarrow \mathrm{X_0}$
b. $\mathrm{X}' \rightarrow \mathrm{X_0\ YP}$

Thus, when the parser has scanned the head ($\mathrm{X_0}$), it will be in the following state. (We use the dotted item notation, illustrated in Appendix A.3.)

(25) a. $\mathrm{X}' \rightarrow \mathrm{X_0} \bullet$
b. $\mathrm{X}' \rightarrow \mathrm{X_0} \bullet \mathrm{YP}$

In an LR parser, this situation is a *shift/reduce* conflict. According to the portion of the rule scanned so far, there are two licit next actions: YP could be shifted onto the stack ((25)b) , or $\mathrm{X}' \rightarrow \mathrm{X_0}$ could be reduced ((25)a). The choice of the correct action depends on the category and subcategory of the current token. Thus, if the current token is an intransitive verb, then only (24)a can apply, thus a reduction will be performed. On the other hand, $\mathrm{I_0}$ obligatorily requires that a VP be a continuation in the same rule, thus (24)b can apply.

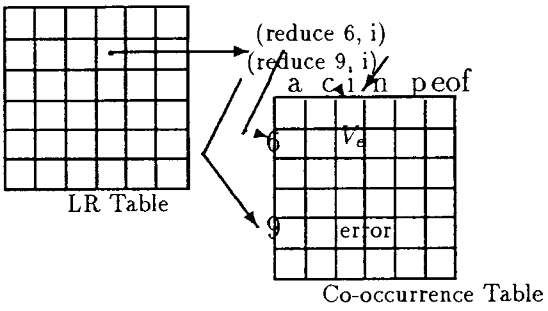


Figure 3.2
Interaction between LR Table and Co-occurrence Table

The choice of possible next states is restricted to one, in most cases, by comparing the options compiled in the LR table, which are deduced exclusively from the $\overline{X}$ grammar, and the prediction of the co-occurrence table, which depends on the actual input token. The way this prediction is accessed in the parser is shown visually in Figure 3.2. Whenever necessary, the parser can consult a table of "compatible next tokens" to decide quickly whether a reduction can be applied. If no reductions are compatible with the current token and the next, then the next token is shifted.

To illustrate, figure 3.3 presents several cases of possible continuations in the context of head movement. They are all triggered by the configuration which is called **unary_head**, namely when a head, such as I_0 or V_0 moves into the location of another head. The first case occurs when the infinitival marker, or a modal verb is the current token, which is directly projected to I_0, then the following category must be a verb, consequently reductions of an empty head or an empty maximal projection are ruled out. The second case illustrates *do* support, *e.g.* the usage of *do* in the position of C_0 in questions, such as *Who do you like?*. Again, the only possible continuation is an NP. For declarative main verbs, we can consult the subcategorization frame of the verb to disambiguate whether they should be followed by an empty maximal projection, if they are transitive, or if the following token, which could be a PP, an adverb, or simply the end of input marker should be shifted onto the stack, because the verb is intransitive.

$\overline{X}$	Next Token						
	XP	NP	$to+V$	V	$P+N$	adv	$\$$
$I_0 \downarrow W_0$				V			
$C_0 \downarrow V_0$		NP					
$V_0 \downarrow W_{0_{intr}}$					PP	adv	$\$$
$I_0 \downarrow V_{0_{trans}}$		NP			NP_e	NP_e	NP_e

Figure 3.3
Co-occurrence Table for Unary Head Production

The co-occurrence table is also useful to reduce nondeterminism, when in need to encode the type of complement phrase that follows a particular verb. The category of the sentential complement of a verb cannot be stored in the phrase structure rules, since only $\overline{X}$ rules are used.

(26) VP → V CP
VP → V IP

Rules such as (26) are needed because some verbs, for instance, raising verbs such as *seem, appear* are subcategorized for a CP complement which can optionally be deleted in infinitivals, as is shown in (27).

(27) a. John seems [$_{IP}$t to like Mary]
b. It seems [$_{CP}$that [$_{IP}$John likes Mary]]

Optional CP deletion is a lexical property, whichmany other verbs do not have. Such verbs, instead, license an empty complementizer, as shown in (28).

(28) a. John thinks that Mary is sick
b. John thinks $\emptyset$ Mary is sick

Encoding categorial restrictions in the table of co-occurrences enables the parser not to engage in useless computation. Thus, for example, when parsing the raising verb *seem* and the next token is the infinitival marker, then the parser knows it must insert a trace before shifting the infinitival marker *to* onto the stack. In the case of the verb *think* the empty complementizer is inserted if the following word is an NP.

The Role of Syntactic Features

It is important to note that, even if a co-occurrence table is used, which encodes categorial information, syntactic features such as ±Case still have an independent role in this parser, as predicted by our classification in IC Classes above. For example, GB theory analyzes passive sentences as having an empty category after the verb, from which the passive subject has moved, for instance, *John is loved t.* The co-occurrence table predicts that after a transitive verb there should be an object, but it does not contain information about Case or θ-assignment capabilities of verbs, and thus does not use any of this information. This information is encoded in the lexicon, and not reduplicated in the co-occurrence table. It is however necessary, and it is done on line, to determine the feature assignment of moved elements. For instance, if an empty category is posited as the object of a verb, then the empty category must be able to unify with a possible antecedent, with which it shares Case and θ features.

3.3 COMPACTNESS OF THE DATA STRUCTURES

The organization of the parser presented so far is computationally advantageous. Consider again the $\overline{\text{X}}$ grammar that we use in the parser, shown in Figure 3.1. One of the crucial features of this grammar is that the nonterminals are specified only for level and headedness. This version of the grammar is a recent result. In previous implementations of the parser, the projections of the head in a rule were instantiated. (See also Berwick (1991a), who suggests a similar kind of partial instantiation, as a possible way of representing $\overline{\text{X}}$ theory.)

I thought that specifying the category label would reduce nondeterminism in the compiled parse table, the rationale being that addition of information could only reduce ambiguity. Such a move was unjustified, however, and contradicted our definition of efficient compilation as compilation of dependent modules.

In fact, the prediction made by the ICMH is that compiling together $\overline{\text{X}}$ theory and categorial information is going to increase the size of the grammar without producing any reduction in the non-determinism contained in the grammar, because category/subcategory information belong to a different IC Class than structural (i.e. $\overline{\text{X}}$) information. This hypothesis was tested by comparing indices of ambiguity across different grammars.

3.3.1 Method and Materials

The size of the grammar is measured as the number of rules or number of states in the LR table. The amount of non-determinism is measured as the average

1	s	$\rightarrow$ x2;
2	x2	$\rightarrow$ y2 x1;
3	x2	$\rightarrow$ x1 y2;
4	x2	$\rightarrow$ y2 x2;
5	x2	$\rightarrow$ x2 y2;
6	x1	$\rightarrow$ x0 y2;
7	x1	$\rightarrow$ y2 x0;
8	x1	$\rightarrow$ x1 y2;
9	x1	$\rightarrow$ y2 x1;
10	y2	$\rightarrow$ x2
11	y2	$\rightarrow$ w2
12	y2	$\rightarrow \epsilon$
13	x0	$\rightarrow$ w0
14	x0	$\rightarrow \epsilon$
15	x2	$\rightarrow$ x1;
16	x1	$\rightarrow$ x0;

Figure 3.4
Grammar 1, which encodes $\overline{\text{X}}$ theory directly

number of conflicts (the ratio between the number of actions and the number of entries in a table.) [2]

Three grammars were constructed, constituting (pairwise) a close as possible approximation to minimal pairs (with respect to IC Classes). They are shown in Figure 3.4, Figure 3.5, and Figure 3.6. Grammar 1 differs minimally from Grammar 2, because each head is instantiated by category. The symbol YP stands for any maximal projection admitted by linguistic theory. Grammar 3 differs minimally from Grammar 2, because it also includes some subcategorization information (such as transitive, intransitive, raising), and some co-occurrence restrictions and functional selection. Moreover, empty categories are "moved up", so that they are encountered as high in the tree as possible. These three grammars are then compiled by the same program (BISON) into three (LA)LR tables. The results are shown in Table 3.1, which compares some of the indices of the non-determinism in the grammars to its size, and Table 3.7, which shows the distribution of actions in each of the grammars.

[2]The average number of conflicts in the table gives a rough measure of the amount of non-determinism the parser has to face at each step. However, it is only an approximate measure for at least two reasons: taking the mean of the conflicts abstracts away from the size of the grammar, which might be a factor, as the search in the table becomes more burdensome for larger tables; moreover, it does not take into account the fact that some states might be visited more often than others during an actual parse.

1	s	→ c2	18	a1	→ a0 y2	35	i1	→ i0
2	c2	→ y2 c1	19	a1	→ y2 a1	36	v2	→ v1
3	c1	→ c0 y2	20	a1	→ a1 y2	37	v1	→ v0
4	c1	→ y2 c1	21	a2	→ y2 a2	38	a2	→ a1
5	c1	→ c1 y2	22	p2	→ y2 p1	39	a1	→ a0
6	c2	→ y2 c2	23	p1	→ p0 y2	40	p2	→ p1
7	i2	→ y2 i1	24	p1	→ y2 p1	41	p1	→ p0
8	i1	→ i0 y2	25	p1	→ p1 y2	42	d2	→ d1
9	i1	→ y2 i1	26	p2	→ y2 p2	43	d1	→ d0
10	i1	→ i1 y2	27	d2	→ y2 d1	44	y2	→ ϵ
11	i2	→ y2 i2	28	d1	→ d0 y2	45	Y2	→ n2
12	v2	→ y2 v1	29	d1	→ y2 d1	46	y2	→ c2
13	v1	→ v0 y2	30	d1	→ d1 y2	47	y2	→ i2
14	v1	→ y2 v1	31	d2	→ y2 d2	48	y2	→ v2
15	v1	→ v1 y2	32	c2	→ c1	49	y2	→ a2
16	v2	→ y2 v2	33	c1	→ c0	50	y2	→ p2
17	a2	→ y2 a1	34	i2	→ i1	51	y2	→ c2
						52	y2	→ d2

Figure 3.5
Grammar 2, where heads are instantiated by lexical categories

	NB OF ENTRIES	NB OF ACTIONS	NB OF RULES	AVERAGE CONFLICTS
GRAMMAR 1	63	123	16	1.95
GRAMMAR 2	793	1319	51	1.78
GRAMMAR 3	251	962	41	3.83

Table 3.1
Comparison of the 3 grammars (compiled into LR tables)

	NUMBER OF ACTIONS													
ENTRIES	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GRAMMAR 1	38	6	8	8	1	2								
GRAMMAR 2	465	68	168	6	42									
GRAMMAR 3	144	43			3	8	5					4	30	14

Table 3.2
Number of actions in the 3 LR tables

1	s	→ i2
2	s	→ c2
3	c2	→ y2 c1
4	c2	→ c1
5	c2	→ ϵ
6	c1	→ c0 i2
7	c1	→ c0
8	c0	→ c
9	c0	→ ϵ
10	i2	→ y2 i1
11	i2	→ i1
12	i2	→ ϵ
13	i1	→ i0 v2
14	i1	→ i0
15	i0	→ i
16	i0	→ ϵ
17	v2	→ y2 v1
18	v2	→ ϵ
19	v2	→ v1
20	v1	→ v0int p2
21	v0int	→ vint
22	v0int	→ ϵ
23	v1	→ v0t n2
24	v0t	→ vt
25	v0t	→ ϵ
26	v1	→ v0rais i2
27	v0rais	→ vrais
28	v0rais	→ ϵ
29	v1	→ v0int c2
30	y2	→ c2
31	y2	→ i2
32	y2	→ n2
33	y2	→ v2
34	y2	→ p2
35	y2	→ ϵ
36	n2	→ n
37	n2	→ ϵ
38	p2	→ p0 y2
39	p2	→ ϵ
40	p0	→ p
41	p0	→ ϵ

Figure 3.6
Grammar 3 contains information about functional selection and co-occurrence restrictions

3.3.2 Discussion

Consider Grammar 1 and Grammar 2 in Table 3.1. Grammar 2 has a slightly smaller average of conflicts, while its number of rules increases 3 times and the number of entries increases 12 times, compared to Grammar 1. This is the predicted expansion of the grammar without decrease in non-determinism. As the number of rules is expanded, but no "filtering" constraint is incorporated in Grammar 2 with respect to Grammar 1, this result might not seem surprising.

However, the ICMH is also confirmed by the other pairwise comparisons and by the global results. Grammar 3 has a higher number of average conflicts than Grammar 2, but it is smaller, both by rules and LR entries, so it is more compact. Notice that adding information (subcategory, selection, etc.) has a filtering effect, and the resulting grammar is smaller. However, adding information does not reduce non-determinism. Compared to Grammar 1, Grammar 3 does not show any improvement on either dimension. Grammar 3 is both larger (4 times as many LR entries) and more non-deterministic than Grammar 1. Globally, one can observe that an increase in grammar size, either as a number of rules or number of LR entries, does not correspond to a parallel decrease in non-determinism.

As Table 3.2 shows, the distribution of the conflicts in Grammar 3 presents some gaps. This occurs because certain groups of actions go together. One mainly observes two patterns of conflicts. In those states that have the highest number of conflicts, all rules that cover the empty string can apply, while in those states that have an intermediate number of conflicts, only those rules can apply that have a certain $\overline{X}$ projection level, and that cover the empty string (e.g. all XP's, independent of category, that cover the empty string). An example of conflicts is given in Figure 3.7. So, for instance, all X_0 projections present the same `(reduce 33),(reduce,44)` conflict. This observation confirms that categorial information does not reduce non-determinism, but rather it multiplies out with structural configurations. Even introducing "filtering" lexical information (co-occurrence restrictions and functional complementation) does not appear to help. In fact, patterns of ambiguities caused by empty categories occur according to structural partitions.

One can conclude that the qualitative observation supports the numerical results: Introducing categorial information is not advantageous, because it increases the size of the table without decreasing significantly the average number of conflicts. Moreover, by using a category-neutral grammar, the number of rules that are compiled in the LR table is very small. Using LR compila-

```
ytab(1,X,Actions):-ytab1(X,Actions).
ytab1($,[(reduce,33),(reduce,44)]).
ytab1(a0,[(reduce,33),(reduce,44),(shift,5)]).
ytab1(a1,[(shift,15)]).
ytab1(a2,[(shift,14)]).
ytab1(c0,[(reduce,33),(reduce,44),(shift,1)]).
ytab1(c1,[(shift,9)]).
ytab1(c2,[(shift,21)]).
ytab1(d0,[(reduce,33),(reduce,44),(shift,6)]).
ytab1(d1,[(shift,19)]).
ytab1(d2,[(shift,18)]).
ytab1(i0,[(reduce,33),(reduce,44),(shift,2)]).
ytab1(i1,[(shift,11)]).
ytab1(i2,[(shift,10)]).
ytab1(n2,[(reduce,33),(reduce,44),(shift,7)]).
ytab1(p0,[(reduce,33),(reduce,44),(shift,4)]).
ytab1(p1,[(shift,17)]).
ytab1(p2,[(shift,16)]).
ytab1(v0,[(reduce,33),(reduce,44),(shift,3)]).
ytab1(v1,[(shift,13)]).
ytab1(v2,[(shift,12)]).
ytab1(y2,[(shift,22)]).
ytab1(_,[(reduce,33)]).
```

Figure 3.7
Example of Conflicts in the Table compiled from Grammar 2. The table entry for state 1 is shown (indicated by ytab1). $ as usual indicates end of input. a0,a1,a2 indicate the X_0, X', X'' projections for the category adjective.

tion techniques then becomes feasible. It is well-known that these compilation techniques can generate parse tables of thousands of states, if applied to an ordinary context-free grammar, with clear problems for maintenance. From the linguistic point of view, a parse table with so many states would miss the generalizations about interleaving of configurations and filtering constraints.

	NB OF ENTRIES	NB OF ACTIONS	NB OF RULES	AVERAGE CONFLICTS
GRAMMAR 1	19	62	16	3.26
GRAMMAR 1′	19	46	13	2.42
GRAMMAR 2	112	255	51	2.28
GRAMMAR 3	144	368	41	2.62

Table 3.3
Comparison of the 3 grammars (compiled into LL tables)

3.3.3 Extending the test to other compilation techniques

The effects discussed in the previous section could be an artifact of the compilation technique. In order to check that this is not the case, the same three grammars were compiled into LL and Left Corner (LC) Tables. In the LL algorithm, the mother node is recognized top-down before its children, and the parser builds a leftmost derivation; the LC algorithm combines the bottom-up recognition of the extended left-corner of each rule to the top-down recognition of the remaining part of the righthand side of the rule. By comparing the results obtained in the LR, LL and LC compilations, we can compare the effects of varying the order in which grammatical and input information is used.

LL compilation: Discussion

Results similar to the LR compilation, although less clear cut, seem to hold also when one considers the LL compilation method, thus confirming the intuition that they reflect some structural property of the grammar, and are not an artifact of the LR compilation.

The results of the compilation of the same grammars into LL tables is shown in Table 3.3. Grammar 1′ is a modified version of Grammar 1, without adjunction rules. These figures show that there is no relation between the increased specialization of the grammar and the decrease of non-determinism. One can observe that, differently from the compilation in Table 3.1, the LL compilation does not maintain the paired rankings of actions and rules. So, for the LL Table, the co-occurrence of lexical categories does not play a filtering role.

	NB OF ENTRIES	NB OF ACTIONS	NB OF RULES	AVERAGE CONFLICTS
GRAMMAR 1	49	136	16	2.77
GRAMMAR 2	1456	4030	51	2.76
GRAMMAR 3	398	610	41	1.53

Table 3.4
Comparison of the 3 grammars (compiled into LC tables)

Globally, there appears to be an inverse relation between the size of the grammars as number of rules and the average number of conflicts: the larger the grammar the smaller the number of conflicts. This might make one think that there is after all some sort of relation between grammar size and non-determinism. However, this is not true if we look at the number of entries as the relevant measure of size. Moreover, if one looks at Grammar 1′, which is smaller than Grammar 1, one can see that the average number of conflicts decreases quite a bit. This confirms a weaker hypothesis, which is nonetheless related to the initial one, namely that non-determinism does not vary in an inverse function to "content of information".

Some qualitative observations might help clarify the sources of ambiguity in the tables. In all three grammars, the same ambiguities are repeated, for each terminal item. In other words, all columns of the LL table are identical (with the exception of cell [X0, wp] in Grammar 1.) This suggests that lexical tokens do not provide any selective information. Moreover, analogously to the LR tables, projections to the same level have the same pattern of conflicts (multiplied out by category in Grammar 2, for example.) [3]

LC-compilation: Discussion

The same three grammars were compiled in left corner (LC) tables. The result of the compilation are shown in Table 3.4, and the distribution of the conflicts is shown in Table 3.5. As can be seen from Table 3.4, Grammar 2 is 3 times larger than Grammar 1 and it is compiled in a table that has 29 as many entries, but the average number of conflicts is not significantly smaller.

[3]In all cases, this is caused by the $\overline{X}$ form of the grammar. Namely, the loci of recursion and gapping are at both sides of the head, and anything can occur there. Eliminating this property would be incorrect, as it would amount to eliminating one of the crucial principles of GB, namely *move-α*, which says that any maximal projection or head can be gapped.

	NUMBER OF ACTIONS								
ENTRIES	1	2	3	4	5	6	7	18	19
GRAMMAR 1	4	18	15	9	3				
GRAMMAR 2	602	702	48					96	8
GRAMMAR 3	282	92		4	4	4	12		

Table 3.5
Number of actions in the 3 compiled LC tables

The interpretation of the LC table derived from Grammar 3 poses a problem for the ICMH. Compared to Grammar 1, Grammar 3 is bigger, as it contains category and some co-occurrence information, but its average of conflicts is smaller. Thus it seems that adding information reduces non-determinism. On the other hand, compared to Grammar 2, both the table and the average number of conflicts are smaller. Thus, the ICMH is confirmed only by a global assessment of the relation between the content of information and the average conflicts, but not by pairwise comparisons of the grammars. Notice however, that the difference in the two pairwise comparisons confirms that simple categorial information does not perform a filtering action on the structure, while lexical co-occurrence does. This is precisely what I propose to compile in the lexical co-occurrence table.

The qualitative inspection of the tables confirms the clustering of conflicts which is suggested by Table 3.5. Grammar 1 and Grammar 2 show the same patterns of conflicts as the LR and LL tables: conflicting actions cluster with the bar level of the category. So, for example, in Grammar 2 one finds that when the left corner is a maximal projection the action is unique, while when the corner is a bar level projection there are multiple actions and they are the same, independently of the input token. In Grammar 3 the same patterns of actions are repeated for each left corner, independently of the goal or of the input token.

Globally, then, the qualitative inspection of the compiled tables is coherent across compilation methods and appears to support the ICMH, as the interaction of structural and lexical information is the cause of repeated patterns of conflicts. Quantitatively, the results, which are very suggestive in the LR compilation, are less clear in the other two methods. However, in no case they clearly disconfirm the hypothesis. I conclude that categorial information should be factored out of the compiled table and separate data structures should be used.

3.4 PSYCHOLINGUISTIC SUPPORT

I turn now to discussing the properties of the structure building algorithm in the light of the known evidence on the interaction between categorial and structural information.

The parser presented here has the following properties:

- It compiles $\overline{X}$ information separately from categorial co-occurrence information
- It consults these two tables on-line, interleaving them at each input token if necessary
- It uses one token lookahead to compile the LALR table, but it has no lookahead facility at run-time.

The previous section has shown that efficiency arguments based on compactness of data structure support the separation of structural from co-occurrence information.

Parsing by using two different parse tables, which contain different types of information, makes specific predictions about the on-line availability of information to the human sentence processor. In particular, it claims first that the interleaving of structural and categorial (i.e. lexical) information is done at each input token. Consequently, it predicts that subcategorization information of verbs is used immediately. In fact, more specifically, subcategory information is used to disambiguate otherwise structurally ambiguous input. Second, the organisation of data structures that we propose predicts that if the LR table does not have a conflict at a certain point in the parse, there will be no apparent difficulty. Finally, it predicts that, if necessary, the parser can consult the co-occurrence table directly.

On the Use of Subcategory

The token-by-token interleaving of the two tables can be interpreted in the light of the present debate on the use of subcategorization. Consider the following sentences (adapted from Frazier and Rayner (1982)):

(29) a. Sherlock Holmes didn't suspect the beautiful young countess.
b. Sherlock Holmes didn't suspect the beautiful young countess was a fraud.

The verb *suspect* subcategorizes both for an NP and a clausal complement, thus an NP following the verb is locally ambiguous as a potential object of the verb, as in the first sentence, or as the subject of a following clause, in the second sentence.

This kind of local ambiguity has been studied in several experiments. Contradictory results have been found. Some results support a model of sentence processing which is driven by the phrase structure rules of the grammar, and, therefore, does not use word-specific lexical information, but only syntactic information (Frazier and Rayner 1982; Ferreira and Clifton 1986; Mitchell 1987; Rayner and Frazier 1987; Ferreira and Henderson 1990). Other results support a model which assumes that lexical information is used as soon as possible and that parsing is driven by lexical expectation (Ford, Bresnan, and Kaplan 1982; Mitchell and Holmes 1985; Holmes, Stowe, and Cupples 1989; Trueswell, Tanenhaus, and Kello 1993).

Three experiments have supported the theory of phrase-structure-driven processing (Frazier and Rayner 1982; Ferreira and Henderson 1990; Rayner and Frazier 1987). They test the hypothesis that the human sentence processor adopts general parsing strategies, based on economy of syntactic representation. More precisely, the parser attempts to build the representation with the smaller number of nodes (*Minimal Attachment*). When possible alternatives have the same number of nodes, an overriding strategy of *Late Closure* operates, since the parser has the tendency to incorporate incoming material into existing nodes.

In an eye-tracking study, Frazier and Rayner (1982) find increased reading times at the disambiguation region for the reduced clauses (b examples above), which they interpret as evidence of a complexity effect, caused by the garden path in the conditions where the ambiguous NP is not the object. These results are confirmed by Ferreira and Henderson (1990), where the NP/Clause ambiguity is studied. Although they explicitly control for the bias of the verbs, following the norms of Connine, Ferreira, Jones, Clifton, and Frazier (1984), they still find evidence in favour of a Garden Path model of parsing, namely no interaction with verb type. Rayner and Frazier (1987) test a corollary of Minimal Attachment: if the increased reading times are an effect of reanalysis at the disambiguating region, then the clausal complement with an overt complementizer should be parsed faster since it is not ambiguous, as also Holmes, Kennedy, and Murray (1987) note. They find a difference in reading times, with the unambiguous sentence being parsed faster.

The opposite position supports a view where both syntactic class and word-specific lexical information is used, so that a particular preference for a given continuation depends on the verb usage. This model, which has been called the Lexical Guidance Model (Ford, Bresnan, and Kaplan 1982; Mitchell and Holmes 1985; Holmes, Stowe, and Cupples 1989; Trueswell, Tanenhaus, and Kello 1993), predicts a tendency to assemble an NP following a verb as the direct object only if the verb itself is biased for an NP continuation, with consequent need for reanalysis, if the sentence being parsed is in fact a sentence with a reduced clausal complement, as (29)b above. On the other hand, if the verb is biased for a clausal continuation, the analysis of sentences such as (29)b should not show any disruption.

Two studies have found an interaction between verb type and processing time (Holmes, Stowe, and Cupples 1989; Trueswell, Tanenhaus, and Kello 1993). These experiments found an increase in processing times only for NP-bias verbs, but not for S-bias verbs, thus suggesting that the parser uses lexical information immediately. The experiments confirm the hypothesis that subcategorisation becomes available right after the verb is recognized, and that individual lexical biases of verbs influence structural attachments. [4]

In conclusion, it appears that subcategorization is used immediately, although some of the results need to be refined.(See also Gorrell (1991), that shows that

[4]Of course, the results that led to the Garden Path position must be explained. Notice that a simple frequency account is not tenable. Although it is true that the predictions of the Garden Path Theory are empirically indistinguishable, in the reported experiments, from a general preference for direct object attachment of an NP following a verb. Merlo (1994) discusses corpus counts which show that all experiments were rather well-balanced, with the possible exception of the Late Closure experiment in Frazier and Rayner (1982). Even in cases in which the stimuli were strongly biased against a Minimal Attachment continuation, complexity effects were found. Thus the frequency hypothesis which has been proposed by the supporters of the Lexical Guidance Model is not tenable in its simplest form, as it would not explain the findings of the Garden Path experiments.

the results in Mitchell (1987), that supported an extreme version of the garden path model, might be an artifact of the experiment.) These experimental results support the use of lexical information that we suggest.

On the Use of Lookahed

In the present parser a kind of lookahead is used, which might be considered too powerful in the light of the current psycholinguistic evidence. It is now widely accepted that humans do not use a lookahead device, otherwise they woud not experience garden paths in most instances of short strong garden paths. However, the use of one token lookahead at compile time, when compiling the $\overline{X}$ grammar into an LALR(1) is a much weaker mechanism than using one token of input, and all the information it can provide, as a disambiguating device.

An LALR(1) parsing table is a parsing table that has been compiled by using one token look-ahead in the grammar, in this instance the $\overline{X}$ grammar. If the grammar is not LALR(1), the table entries will have multiple actions.

We can test the adequacy of the particular parser we propose, by constructing a grammar, and then checking that for the garden path sentences which could be disambiguated by one token lookahead, the parser presents conflicts at the same point in which the human sentence processor garden-paths. For example, in *The horse raced past the barn fell*, we want to check that the LALR(1) table has a conflict in the state which corresponds to the input token *raced.*

The test is simple. We have constructed an (overdetermined) grammar, in which all the categories are instantiated, which could parse the three sentences in (30). The grammar is shown in 3.8. This grammar is then compiled into an LALR(1) table.

(30) a. Mary expected Eric to leave
b. While she mended the sock fell off her lap
c. The horse raced past the barn fell

Even if the parse table is compiled by using one token look-ahead, the trace of the parse for these sentences shows that there are conflicts at the state where the parse must attach *Eric* and *The sock*, and when it must decide whether *raced* is part of a main sentence or rather of a reduced relative clause. Since no other look-ahead is used, we can conclude that using an LR table is plausible.

1	IP	$\rightarrow$NP I1	11	NP	$\rightarrow$det n
2	IP	$\rightarrow$CP IP	12	NP	$\rightarrow$n
3	IP	$\rightarrow$NP to VP	13	NP	$\rightarrow$NP CP
4	I1	$\rightarrow$I0 VP	14	NP	$\rightarrow\epsilon$
5	I0	$\rightarrow\epsilon$	15	CP	$\rightarrow$OP C1
6	VP	$\rightarrow$v PP	16	CP	$\rightarrow$C1
7	VP	$\rightarrow$v NP	17	C1	$\rightarrow$C0 IP
8	VP	$\rightarrow$v	18	C0	$\rightarrow$while
9	VP	$\rightarrow$v IP	19	C0	$\rightarrow\epsilon$
10	PP	$\rightarrow$p	20	OP	$\rightarrow\epsilon$

Figure 3.8
LALR Grammar

3.4.1 On the Interaction between Lexical and Structural Ambiguity

Categorial Ambiguity with Structural Difference

Experimental evidence on the resolution of categorial ambiguity is naturally explained if a parser is posited that separates structural from lexical information, and where information about lexical co-occurrence is organised independently of structural information and can be accessed directly.

Frazier and Rayner (1987) report on a reading experiment in which they record eye-movements. The experiment is designed to test what mechanism is used to resolve categorial ambiguity. Three hypotheses are compared. The first hypothesis assumes that categorial ambiguity is resolved with a mechanism similar to the one for syntactic ambiguity, namely by choosing a first analysis, which is then discarded if it turns out to be wrong, causing backtracking. The second option would take into account the fact, which has been shown independently (Seidenberg, Tanenhaus, Leiman, and Bienkowski 1982; Swinney 1979), that in retrieving lexically ambiguous items, all entries of the item are accessed for a short time. It could be hypothesized that the parser pursues all analyses in parallel. Finally, a third hypothesis rests on the observation that lexical ambiguity (unlike syntactic attachment ambiguities) is usually resolved in a very local environment, within 2 or 3 words. Thus the parser would delay building structure and look ahead for disambiguating material. A pair of locally ambiguous and unambiguous sentences used in the experiment are shown in (31).[5]

(31) a. The warehouse FIRES *numerous employees each year.*
b. This warehouse fires *numerous employees each year.*

The results of the experiment show that processing times are shorter for the two ambiguous target words than for the corresponding words in the unambiguous sentences. Moreover, processing time of the disambiguating region in the ambiguous condition is more than in the unambiguous one, *i.e.* there is a slowdown. No main effect due to category difference was observed for ambiguous conditions, while consistently longer times for the A-N group were found in the unambiguous conditions. The disambiguating effect was triggered immediately, when the first nonambiguous token (*numerous*) was encountered.

These results are compatible only with the delay hypothesis, as the first analysis hypothesis predicts no processing difference between the unambiguous control and the preferred reading (the N-V interpretation), while the parallel hypothesis predicts longer processing time in the ambiguous portion. More specifically, the speed up in the ambiguous portion is predicted by the delay hypothesis because a costly operation such as structure building is suspended, while the slow down in the disambiguating portion can be attributed to the fact that structure building is resumed. The slower processing for the A-N sequence in the unambiguous sentences is explained by the fact that agreement disambiguates this sequence earlier than the N-V sequence.

These results show that the parser does not use several tokens of lookahead (Marcus 1980; Fong 1991), or parallel computation, as such parsers would be able to solve the ambiguity without disruption. On the other hand, a lexically driven parser (Frank 1992; Abney 1989) could not recover the necessary information to disambiguate the sentence. As structural information is encoded in the lexicon, these parsers would do an exhaustive search for all the possible continuations of all the category labels that can be attributed to the current linguistic item.[6]

[5]The sentences are presented with the addition of typographical conventions which were not used in presenting the material during the experiment. The ambiguous part is shown in all capitals and the disambiguating portion is shown in italics. Frazier and Rayner (1987) also argue that the stress pattern of the sentences with the A-N sequence is such that these sequences cannot be analysed as a compound noun.

[6]Frazier (1989) argues convincingly that these results point away from lexical generation of syntax, such as categorial grammars or, we add, licensing grammars, would imply. She is

A parser that can enter "delay mode" must recognize immediately that a structural ambiguity has been reached. Thus, as a first property, the parser must have some facility to distinguish the inability to assign a successful parse because of structural ambiguity, from, let's say, incorrect feature assignment. The compilation of $\overline{\text{X}}$ rules into an LR table guarantees that the recognition of conflicts will occur at the point of ambiguity, because an LR parser has the valid prefix property. (See discussion in chapter 5.) Moreover, for the delay parser not to stall, it must be able to trigger some alternative routines, to suspend structure building, but continue some other kind of computation that will resolve the structural ambiguity.

Consider what happens in parsing the sentence in (31)a. (I assume a slightly simplified grammar for ease of exposition, where non-binary rules are allowed in limited circumstances. Nothing in the discussion that follows hinges on it.) The parser scans the input and builds structure according to the grammar. *The warehouse fires* is represented as the two following rules in the grammar.

(32) a. XP $\rightarrow$ Det Y2 X1
b. XP $\rightarrow$ Det X1

The parser scans *the warehouse* and builds the structure indicated before the •.

(33) a. XP $\rightarrow$ Det • Y2 X1
b. XP $\rightarrow$ Det • X1
c. X1 $\rightarrow$ X0 •

(33) shows that after scanning *warehouse*, the parser must decide which rule to apply. If *warehouse* is an Adjective, then it must be projected to Y2, and the rule (33)a will be applied, (yielding the item XP $\rightarrow$ Det Y2 • X1), which means that the AP has been scanned but the NP is not yet finished. If *warehouse* is a noun, then (33)b will be applied, (yielding the item XP $\rightarrow$ Det X1 •). The

however incorrect, we believe, in concluding that these results cause difficulty for principle-based grammars, such as GB, and principle-based parsing. She takes $\overline{\text{X}}$ theory to mean that the properties of a phrase are projected from the head, and that principle-based parsing is necessarily against grammar compilation. Neither assumption is necessary for a principle-based parser, but they both are for what Frazier has in mind as an example of principle-based parsing, namely Abney (1986), which is a lexically driven parser.

dot at the end of the rule signals that an entire constituent has been seen (the NP), thus the NP constituent is built, and the parser enters the appropriate state to build a VP.

In order to choose between one of these two actions, however, a disambiguating token must be seen. The parser scans ahead: it looks at *fires*, which does not provide any disambiguating material, and then it inspects *numerous*.

When the parser reaches *numerous*, which is unambiguously an Adjective, the parser infers, based on co-occurrence knowledge and preference, that an adjective is a left corner of an NP, which is more often found after a V, hence *fires* is a V, and *warehouse* is a noun. Thus, the parser chooses rule (33)b, it closes the NP and it starts the VP. Note that while this is being done, the input has already been scanned past *numerous*.

This mechanism is simply the reiterated application of consulting co-occurrence of categories, which are independently encoded, without any information of configuration. At this point, the disambiguation of the conflict is achieved, and the portion of input *warehouse fires* is structured into the appropriate constituent. This corresponds to the slow down effect observed in the non-ambiguous region.

In our parser, then. the delay in categorial selection can be simply explained by saying that the algorithm that is used ordinarily to interleave the LR table with the co-occurrence table is suspended, and a different procedure is used, triggered by the lexical ambiguity of the input. Thus, no unnecessary redundancy is introduced, as the data structures and the compilation tables are unaltered.

In a parser that did not separate category from configuration, such table of cooccurrence would have to be stored, creating a redundancy with the LR table.

The facts reported in Frazier and Rayner (1987), about delay of categorial information, show that categorial information and structural information are not compiled into the same table or data structures, since they can be accessed at different times on-line. Thus, they support our approach to $\overline{X}$ parsing, rather than the interpretation put forth in Dorr (1990) or Fong (1991), where the phrase structure rules obey the format dictated by $\overline{X}$ theory, but they are instantiated by category and then compiled off-line. It also supports the token-by-token interleaving of the two tables. A more powerful device would fail to capture the evidence of the delay. For instance, a co-occurrence table that precomputed sequences of 3 words generates the effect that no temporary ambiguity is detected, which would not account for the speed up in parsing time.[7]

Categorial Ambiguity without Structural Difference

Another piece of evidence that is naturally modelled by our design is lexical ambiguity which does not entail structural ambiguity. In a parser that is driven by $\overline{X}$ configurations, like ours, we expect that sentences such as those in (34), from Pritchett (1992), should not cause garden path effects, as they do not correspond to any conflicting actions in the parse table. This is because the only difference, according to Pritchett, is in the category label, while the structure is the same: (34)a shows the auxiliary use of HAVE, while (34)b shows its use as a main verb. No processing breakdown occurs in parsing these sentences.

(34) a. Have the students devoured their dinners?
b. Have the students devoured by the lions

Pritchett (1992) assigns the same $\overline{X}$ configurations to these sentences as shown in Figure 3.9.

Thus we correctly predict that these sentences cause no difficulty, even without use of lookahed, which was instead used by Marcus (1980) to capture these facts. (Pritchett (1992) also discusses small clause constructions under perception verbs and causative constructions.)

Apparent counterexamples are cases in which lexical ambiguity does cause processing breakdown, such as those shown below. (The garden path effect is noted by ?.)

(35) a. ? The prime number few.
b. The prime number seven.

[7]In a recent paper, MacDonald (1994) argues that the unambiguous condition in Frazier and Rayner (1987) could have introduced an extra factor of difficulty, as it used a deictic determiner. If other types of disambigautions are used the speed up in the ambiguous region for the NN interpretation disappears, while it is still present in the NV interpretation. MacDonald (1994) provides an alternative explanation of these facts in a constraint-based framework.

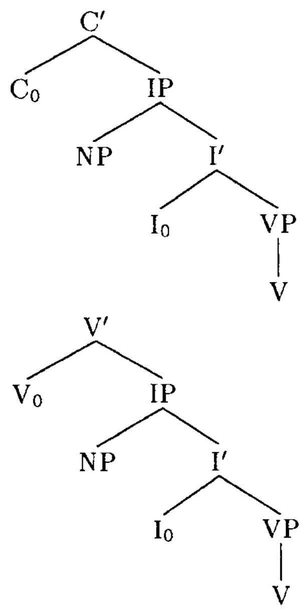


Figure 3.9
The structural representation of the sentences in (34)

(36) a. ? The old train the young.
b. The old train came into town.

At first sight these cases present the same kind of ambiguity as *the warehouse fires*, namely an A-N/N-V ambiguity. Pritchett (1992) however argues convincingly that there is a basic difference between the two. In particular, *the old* which is derived from the adjective *old*, is a zero-level nominal, *i.e.* a nominal that cannot undergo modification, quantification and that must be interpreted as a generic, as the examples in (37) show.

(37) a.?? The ugly old
b.?? All old/ Some old
c.?? The old from New York

As such, Pritchett argues, *the old* is not listed as an independent item in the lexicon, and only the categorization as adjective is at first accessed and incorporated in the structure, giving [$_{NP}$ the old train]. When *the young* is accessed reanalysis must occur. In the instance of *the warehouse fires*, on the other hand, a true lexical ambiguity, listed in the lexicon, causes the parser to behave as described above.[8]

[8]Pritchett (1992), it must be noted, does not accept Frazier and Rayner (1987)'s analysis of such cases. He argues that *warehouse fires* is a compound nominal, thus *warehouse* is univocally accessed from the lexicon as an N, and integrated into the structure. The ambiguity of *fires* as N or V does not cause any processing breakdown because either interpretation can be incorporated into the structure without violating the OLLC. As Pritchett assumes that processing proceeds unimpeded, however, his analysis does not account for the speed up in the ambiguous region found by Frazier and Rayner. Moreover, it incorrectly predicts anomaly effects. For instance, if nominal adjectives are categorized as Ns, and they receive a θ-role as soon as they are incorporated into the structure, as Pritchett assumes, then *Mary talked to the basketball*, presented as a prefix of *Mary talked to the basketball player* should cause an anomaly effect. Intuitive evidence shows that it does not. (I thank Paul Gorrell for this observation.)

4

THE COMPUTATION OF SYNTACTIC FEATURES

4.1 INTRODUCTION

In the previous chapter, I have illustrated how configurations and categorial information are encoded in the parser. They are represented as tables which are consulted on-line by an LR algorithm. Not all the principles of the grammar, however, are represented in these tables. In particular the annotation of the nodes by what I have called *syntactic features*, *e.g.* θ-role and Case is performed on-line, while building the tree. Feature assignment can be strictly local to one of the configurations defined by $\overline{X}$ theory, or it might involve sharing features between distant elements. This kind of feature assignment will be discussed in the first sections.

Features that are not assigned within the maximal projection of the assigning head ($\overline{H}$-local features) encode relations between elements that are more distant from each other than the domain delimited by a maximal projection. In section 4 I discuss the $\overline{H}$-local features that are encoded in *chains*, concluding the discussion of how a parse tree is recovered and annotated, by presenting the algorithms that resolve long distance dependencies, and by discussing some interesting problems related to their efficient and incremental computation. Specifically, this chapter discusses the following issues.

- The method to interleave the structure building rules and the feature annotation: I propose that feature annotation is indexed into configurations of the LR Table.

- The representation of nodes in a tree that support the use of unification, for efficient feature annotation, and the algorithm to annotate nodes, within a local portion of the tree.
- The computation of long distance dependencies. Techniques to assign features efficiently and incrementally in both head-initial and head-final languages are discussed.

On parsing a sentence, more than the mere hierarchical and linear information, encoded in the $\overline{X}$ rules, needs to be recovered. Consider, for example, the pair in (38).

(38) a. John loves [Mary]
b. John talks [to Mary]

The two bracketed complements of the verb are the recipients of the actions of *loving* and *talking*, respectively. However, the speaker's intuition is that they stand in a different relation to the verb. *Mary* in (38)a is the *patient* of the action, while in (38)b *Mary* is less affected by the action. This intuition is expressed by the notion of thematic relation (θ-role). We say that *love* assigns a *patient* role to *Mary*, while *talks* assigns a *goal* θ-role to *Mary*. Moreover, the syntactic structure is not exactly parallel: *loves* assigns a θ-role to *Mary* directly, while *talk* assigns its θ-role indirectly, through the preposition *to*, to *Mary* which is an indirect (or oblique) object. Thus, more abstractly, the relation between a verb and its complement can be captured by assuming that each verb has a certain number of thematic slots, and that each of its arguments fills one.

The thematic relations of verbs are expressed by linear order in English, and (39)a is not equivalent semantically to (39)b.

(39) a. John loves Mary
b. Mary loves John

Other languages, however, grammaticalize thematic relations differently. Thus in Latin, (40)a and (40)b are equivalent because the concepts of *agent* and *patient* are expressed by the case endings *-us* and *-em*, respectively.

(40) a. Caesarem Brutus necavit
b. Brutus Caesarem necavit
"Brutus killed Caesar"

The agent receives nominative case, while the patient receives accusative case. Languages like English, that do not show overt case, are hypothesized to have abstract Case.

Current GB theory assumes other features too. For example, $\pm\gamma$. Take the following minimal pair.

(41) a. Who does John say that Mary loves e_i ?
b. * Who does John say that e_i loves Bob?

In the first sentence the object of the verb *love* is questioned, while in the second sentence the subject is questioned. Notice moreover that whatever restriction rules out the sentence in (41)b, it is syntactic, as the meaning of a hypothetical correct sentence is quite clear: *Who is the person x, such that John says that x loves Bob?* GB theory assumes that, after a series of complex vicissitudes, e_i in (41)a receives $[+\gamma]$ as a feature, while e_i in (41)b receives $[-\gamma]$. $[-\gamma]$ means that the empty category could not occur in that position. The assignment of $[\pm\gamma]$ is presided by the Empty Category Principle.

As we see, the parser, besides information related to the configuration of the parse tree, must recover information that expresses these notions, which are needed to interpret the sentence. In the current and the following section the algorithms that are related to feature assignments will be presented. Such assignments fall into two types: local and non-local. Local constraints are expressed as functions over very small subtrees, defined by the sisterhood relation. Non-local constraints span over a bigger portion of the tree.

In both types of assignments, however, another problem must be discussed, which is the problem of how the phrase structure rules and the constraints interact. I start the presentation by abstracting away from the actual content of the constraints, to concentrate on the way they interact with the $\overline{\text{X}}$ rules and the way they are used to reduce the nondeterminism of the modified LR table.

4.2 THE INTERLEAVING OF CONSTRAINTS

By factoring out $\bar{X}$ rules and constraints, one problem arises, which has been called the *principle interleaving problem* (Fong 1991): how is the connection between the rules and the relevant constraints for those rules retained? Three solutions are possible.

- All possible phrase structure rules are built, constraints are applied to a forest of trees. This approach is provably correct, but it is very space intensive, a forest of hundreds of trees can be built for even small grammars, and it is not very explanatory, in that the entire search space is visited.
- All constraints apply at every reduction. Such approach is also provably correct, but it is not explanatory nor efficient, because it applies many constraints in configurations where they are vacuously true.
- Only a subset of the constraints applies to every phrase structure rule.

I adopt the third approach. The parts of GB theory that do not deal with configurational information are implemented in this system as constraints on rule reduction. This technique is borrowed from standard compilation techniques for programming languages, and it has been used by other people, notably by Correa (1988) and Fong (1991). Principles of the theory such as θ-assignment, Case marking, and the formal licensing of empty categories are expressed as conditions which need to be satisfied for the rule to apply.

This organization embodies one of the observations on the structure of linguistic principles that was presented in the introduction: linguistic principles can be seen as conjunctive statements of conditions. I proposed there that the different conjuncts be satisfied separately. The configuration is encoded separately in the LR table, and so is the *substantive* condition, which is encoded in the co-occurrence table. Since the conditions to build structure are expressed as a conjunction of constraints of different types, conditions on rule reduction perform also a restrictive action on the search space of possible configurations, when more than one rule reduction can be applied. For example, consider the entry in the LR table shown in (42).

(42)
```
ytab0(end_of_file,[(reduce,9),(reduce,11)]).
ytab0(s,[(shift,10)]).
ytab0(w0,[(shift,1),(reduce,9),(reduce,11)]).
ytab0(w2,[(shift,2),(reduce,9),(reduce,11)]).
ytab0(x0,[(shift,6)]).
ytab0(x1,[(shift,4)]).
ytab0(x2,[(shift,3)]).
ytab0(y2,[(shift,5)]).
```

When the parser is in this state, several rules can reduce. For example, the first line of (42) shows that the parser is in state 0 and that the input has been expended. In this configuration, two actions are possible: the parser can reduce by rule 9, or it can reduce by rule 11.

Each rule is "hooked" to a set of conditions that must be satisfied for the rule reduction to apply, some of which are shown in Figure 4.1 below. For example, rule number 9 is labelled *complement*. This rule requires that several conditions be satisfied, in order to apply. In particular, the head must assign a θ-role to the following complement, and the categories of the head and the complement must be compatible. If these constraints are satisfied, the structure building rule applies. A side effect of the satisfaction of constraints is that nodes in the tree become annotated.

If the constraints are not satisfied, then the rule cannot reduce, and a nondeterministic choice point is eliminated. If more than one rule can reduce, because more than one set of constraints is satisfied, that means that the sentence is structurally ambiguous (either globally or locally).

4.3 THE ASSIGNMENT OF LOCAL SYNTACTIC FEATURES

Syntactic features are those features that annotate a node in a tree as a consequence of a syntactic relation to another node in the tree. Consider, for instance, the syntactic feature [±barrier]. According to Cinque (1990) and Rizzi (1990) a maximal projection is a barrier, *i.e.* it receives the feature [+ barrier], if it is not directly selected by a category non distinct from [+V]. Thus a node, which is projected from the lexicon without any features relative to bar-

RULE	CONSTRAINTS
specifier	categorial selection predication percolation of features
complement	categorial selection θ-marking case marking isa-barrier percolation of features
modifier	categorial selection percolation of features
adjunct	categorial selection percolation of features
unary xmax	percolation of features
unary head	assignment of features to the specifier percolation of features complement head selection
empty head	categorial selection (ECP) feature percolation
empty xmax	licensing (ECP) locality

Figure 4.1
Interleaving of Rules and Constraints

rierhood, is annotated [±barrier], as a consequence of the relation of categorial selection from a head. For example, in sentence (43)a the maximal projection IP is directly selected by a node which is not of category [+V] (*someone* is a noun), thus it is marked [+barrier], while in (43)b the projection IP is marked [-barrier].

(43) a. * To whom have you found someone [$_{IP}$who would speak t]?
b. To whom do you regret that [$_{IP}$you could not speak t] ?

Assignments of features fall into two types: local to a head and not local to a head. I define them here.

Definition

Inherent Feature

A head *H* which is the immediate projection from the lexicon of token *W* inherits all the lexical features of *W*. Such features are called *inherent*.

I consider X0 the immediate projection of a lexical item W, X′ the immediate projection of X0 and X″ the immediate projection of X′.

Definition

H-Local Assignment

A feature assignment of the feature *F* is H-local iff it occurs within the maximal projection of the head which bears *F* as an inherent feature.

Definition

$\overline{\text{H}}$-Local Assignment A feature assignment of the feature *F* is $\overline{\text{H}}$-local iff it occurs outside of the maximal projection of the head which bears *F* as an inherent feature.

X''	$\rightarrow Y''\ X'$	specification
X''	$\rightarrow X'\ Y''$	
X'	$\rightarrow X\ Y''$	complementation
X'	$\rightarrow Y''\ X$	

Figure 4.2
Case Assignment Configurations

I shall talk of H-local features and $\overline{\text{H}}$-local features, meaning features that are assigned in an H-local configuration or features that are assigned in a $\overline{\text{H}}$-local configuration, respectively. Examples of H-local features, which are assigned within the maximal projection in which they are projected from the lexicon, are θ-assignment, Case assignment, and the feature that signals formal licensing of empty categories, $\pm\gamma$. Examples of $\overline{\text{H}}$-local features are types of coindexations, such as those between empty anaphors (traces of NP-movement, see below) and their lexical antecedent, or intermediate traces in $\overline{\text{A}}$ chains, or antecedent government.

The way in which I treat these two classes of assignments separates the process of actual feature assignment from imposing the *locality* conditions between the links of chains that are related to the $\overline{\text{H}}$-local feature assignments. This partitioning is mirrored naturally by the LR architecture, as I can use ***attributed grammars*** to perform feature assignment, both local and not local, and I can further constrain the $\overline{\text{H}}$-local feature assignment with the appropriate locality conditions. Attribute grammars are discussed in the next section, while locality conditions are discussed in chapter 5

4.3.1 The Linguistic Features

Five syntactic features are crucial to perform syntactic analysis: $\pm$Case, $\pm\theta$-role, $\pm$Referential, $\pm$barrier, $\pm\gamma$.

Case Case assignment regulates the distribution of NPs in the sentence. A $\pm$ Case feature is necessarily assigned upon reduction of a Case assigning head with its complement and of an I$'$ with its specifier. The Case assignment configurations are shown in Figure 4.2. There is no need to impose any restrictions on the categories. The Case assigning ability is determined in the lexicon for each lexical element, paired with the θ-roles.

θ-role θ-roles are needed to recover the predicate-argument structure of the sentence. They are assigned in the same configurations in which Case is assigned. θ-roles are assigned to subjects, if the verb is not unaccusative (Burzio 1986), by percolating the external θ-role of the verb to the I head and to its I′ projection.[1]

Referential [±Referential] is a feature annotating nodes to compute binding (Rizzi 1990) once the tree is built. It is assigned freely at S-structure, but it becomes part of the computation of the ECP at LF. A referential feature implies the existence of a value for the θ-role feature. θ-roles can be referential and not referential. Referential θ-roles are those that denote the participants in an event, such as *agent, patient, goal*. Only verbs that assign a θ-role can assign a Referential θ-role feature. It is however a feature that must be explicitly listed in the lexical entry of the θ-role assigner, as it is not clear if there is a systematic correspondence between the content of the θ-role and its referentiality.

Barrier This feature is needed to characterize the domains of extraction. Barrierhood is determined upon reduction of a head with its complement. It is a lexical feature for the [±barrier] assigner, since it depends on its [+V] distinguishability (Cinque 1990). Hence, a [+V] head assigns [-barrier] to its complement, and a [-V] head assigns [+barrier].

γ This feature expresses the distribution of empty categories in the phrase marker. $\pm\gamma$ is a feature that determines whether an empty category has been licensed or not. Some empty categories receive it at S-structure, if they are governed by an appropriate head, others receive it at LF (Rizzi 1990).

4.3.2 The Implementation

In our representation, each node is a quintuple of the form *(G,Role,Ref,Case,Barrier)*, with the meaning and values in (44).

[1]This holds if I extend the notion of H-local to features assigned within the maximal projection and its functional complex, e.g. V+I+C. Or we could alter the hypotheses about subjects and assume VP internal subjects. I adopt the former hypothesis, but so far, nothing impinges on it.

(44) **G** is the γ feature, which could be either an atomic constant value or unspecified.
Role is the θ-role assigned to the node, which can be either an atomic constant taken from a given set of role labels such as { agent, theme,...} or unspecified.
Ref is the referential feature, which could be either an atomic constant or unspecified.
Case is the Case feature, whose value can either be an atomic constant from a given set of language-specific possible cases or unspecified.
Barrier is the barrierhood value, which could be either an atomic constant or unspecified.

Syntactic features are attached to a node when the node is projected from the lexicon. Each node in the tree has the format *(ID, Lexical_Features, Syntactic_Features)*. Thus, each node is a triple *(ID,L,S)* where ID is an identification number for each node, L is a quadruple of lexical features, and S is a quintuple of syntactic features.

At the beginning of the parse, syntactic features are uninstantiated variables. They acquire values on rule reduction through unification. For instance, θ-roles can be assigned locally, in a sister relation, as shown in (45).

(45) $V' \rightarrow V\ NP$ { θ-assign(V,NP) }

As was noted in chapter 1, the ICMH is less stringent than the requirement of strict transparency for principle-based parsers, because structural licensing information is used as soon as possible, while other types of annotations are performed later on in the parse. This is particularly convenient when postulating empty categories. The formulation of the ECP that is found in Rizzi (1990) only requires proper head government. This condition will be satisfied if the closest head in the minimal domain imposed by Relativized Minimality satisfies a substantive condition, namely a condition on the category label of the head. This is rather simply encoded, as shown in Figure 4.5. The top routine `some_head_can_license(Symbol,SStk)` is called when attempting to postulate an empty maximal projection. Moreover, this kind of feature assignment does not require checking locality conditions, as it is the relation between a head and a maximal projection in the "minimal domain of the head", *i.e.* its own maximal projection. The admissible configurations are already encoded in the

CONSTRAINTS	FUNCTION
head feature percolation	consults co-occurrence table and determines cooccurrence restrictions among heads
feature absorption	absorbs Case and θ-role if verb is passive
θ-marked	marks node with available θ-role
case marked	marks node with available Case
c-select	categorial selection
is-a barrier	checks if maximal projection is a barrier
license empty head	checks features of closest lexical head
license empty Xmax	checks features of closest lexical head

Figure 4.3
Local Constraints

CONSTRAINT	FEATURES
head feature percolation	category { N,V,C,I,...}
feature absorption	case, θ-role, passive
θ-marked	θ-role, $\pm$referential
case-marked	$\pm$case
c-select	category { N,V,C,I,...}
licensed empty xmax	$\pm\gamma$

Figure 4.4
Local Constraints and their Range of Features

$\overline{\text{X}}$ rules that are compiled in the LR table, onto which the H-local conditions apply. In Figure 4.3 and Figure 4.4 I show the content of the features that are manipulated and their values.

```
%==================================================
constraints(empty_xmax,Symbol,SStk,NewStk,Node,C,C):-

      some_head_can_license(Symbol,SStk),
      locality(SStk, Symbol, NewStk).


%==================================================

some_head_can_license(_,[Y2,X0|SStk]):-

      preceding_head_is_v([Y2,X0|SStk],Head),
      nodegrid(Head,Grid),
      available_role(Grid,_).

some_head_can_license(_,SStk):-

      preceding_head_is_not_empty(SStk,Head),
      lexical_node(Head),
      nodegrid(Head,Grid),
      available_role(Grid,_).

some_head_can_license(_,SStk):-

      preceding_head_is_not_empty(SStk,Head),
      functional_selection(Head,Head1),
      preceding_head(SStk,Head1),
      nodegrid(Head1,Grid),
      available_role(Grid,_).

some_head_can_license(Symbol,_):-

      lexical_head(Symbol),
      nodegrid(Symbol,Grid),
      available_role(Grid,_).
```

Figure 4.5
Algorithm to Compute Proper Head Government

4.4 COMPUTING LONG DISTANCE DEPENDENCIES

In this section I discuss the linguistic analysis that is assigned to sentences which contain long distance dependencies, such as questions. Secondly I will illustrate some computational problems that arise in this computation and the solutions adopted here, which are supported by observations on the availability of disambiguating syntactic features, such as Case, in human sentence processing.

4.4.1 Chains: The Linguistic Facts

Informally, a chain is a syntactic object that defines an *equivalence class of positions* for the purpose of feature assignments and interpretation. For example, take the passive sentence in (46).

(46) $Mary_i$ was loved t_i

The sentence in (46) contains the chain *($Mary_i$, t_i)*. Here t_i receives a θ-role from the verb, but no case, which is absorbed by the passive morphology, while $Mary_i$, receives nominative case because it is in a structural position that is inherently case marked, Spec of IP. This position, though, receives no θ-role, because of passive morphology again. The set of positions, however, satisfies the conditions on lexical argument NPs, namely one half of the θ-criterion and the Case Filter. We define a chain and the link in a chain as follows.

Definition

Chain

A chain C, $(E_1, ..., E_n)$ for n $\geq$ 1, is a sequence of elements in a phrase marker, where E_1 is the head of the chain and E_n is the foot of the chain.

Definition

Link

A link of a chain L, (E_i, E_{i+1}) is an ordered pair of consecutive elements of the chain.

It will be useful in what follows to classify the different kinds of chains according to different criteria.

Type

There are different kinds of chains depending on the syntactic status of the moved element, whether it is a head or a maximal projection, whether it is an NP or a *wh*-element. We can distinguish at least four types: *wh*-chains, such as (47), NP-chains, such as (48), head movement chains, shown in (49), chains formed by movement of maximal projections, exemplified in (47) and in (48).

(47) Who_i did John love t_i?

(48) a. $Mary_i$ was loved t_i
b. $Mary_i$ seemed e'_i to have been loved t_i

(49) Gianni ama_i t_i Maria
"John loves Mary "

Landing Site

Chains can also be classified according to the status of the landing site: A chains are those that are headed by an element in A (argument) position, shown in (50); $\overline{\text{A}}$ chains are headed by an element in $\overline{\text{A}}$ (non-argument) position, such as (51) .

(50) $Mary_i$ seemed e'_i to have been loved t_i

(51) Who_i did John think e'_i that Mary loved t_i ?

Multiple Chains

More than one chain can occur in a sentence. Multiple chains occurring in the same sentence can either be disjoint, intersected or composed. Disjoint chains are nested, as in (52).

(52) Who_i did $Mary_j$ seem t_j to like t_i

If chains intersect, they share the same index and they have exactly one element in common. They are always $\bar{A}$ + A, in this order, as shown in (53).[2]

(53) Who_i did you think e'_i t_i seemed t_i to like Mary?

If chains compose they don't have intersecting elements, but they create a new link: if E_n is the foot of one chain and E'_1 is the head of the following chain then $< E_n, E'_1 >$ is a link in the composite chain. This is exemplified in (54).

(54) Who_i did you meet t_i O_i without greeting t_i ?

4.4.2 The Representation of a Chain

In this parser, a chain is represented as a triple (P,L,S) where

P is a list of the positions in the tree that constitute the chain

[2]The reverse order would be the trace of improper movement, because one of the intermediate traces would be in $\bar{A}$ position, but bound by the A head of the A chain, thus violating principle C of binding theory. For instance, **Who$_i$ e$'_i$ seems t$_i$ that Mary like t$_i$?* is an example of improper movement.

L are the lexical features that belong to the head of the chain and thus to the whole chain, since only one element in a chain can be lexically realized.

S are the syntactic features of the chain. In this discussion, only three of the syntactic features assigned to each node are relevant for chains: Case, Role, Ref.

This representation captures the restrictive interpretation of a chain as *an equivalence class of positions*, because conditions can only be imposed on the chain and not on the single nodes that compose it. As such, it differs from those representations of chains as linked lists of nodes in a tree structure, where each node could be independently specified (Clark 1990; Correa 1991). This choice of representation is parsimounious: by using a triple as the representation of the chain, it is unnecessary to re-encode precedence information about all the processed input that is contained in the stack, or structural information that is contained in the tree. The chain shares elements with the stack of the parser, so that updates to the stack are propagated to the chain and vice versa. Moreover, it is more appropriate descriptively, as chains are represented as properties of a whole sentential phrase marker, and not of a single node (differently from (Correa 1988; Correa 1991), see below). Each sentence is assigned two lists of lists, one for A chains and one for $\overline{\text{A}}$ chains, thus, in principle, an unbounded number of chains for each sentence is possible.

The most typical property of a chain is uniqueness of feature annotation. Only one θ-role and one Case can be assigned to each chain, and only one element in the chain can be lexical. In order for this uniqueness property to obtain I define *unique term unification* as a mechanism for feature assignment. In unique term unification I require that at most one of the elements involved in the operation of unification is instantiated. The semantics of the operation is shown in Figure 4.6.

We can observe that the uniqueness restriction is a kind of derivative encoding. It encodes that only one assignment can occur for each type of feature to a given chain, even if the new assignment does not add any new information. In fact, it is a strict form of information monotonicity, where every assignment must add information.

It is not clear why it should be so, but uniqueness seems a pervasive modality of feature assignment. While phrase structure appears to be built under the monotonicity restriction, feature assignment seems to obey uniqueness. We

Given two terms A and B, `unify(A,B,C)` iff A and B are terms
that encode syntactic features attached to a node or attached to a chain,
and C is their most general unifier under *unique unification.*

```
unify(A,B,C) iff A is a variable or B is a variable.

unify(A,B,C) iff unique_unify(case(A), case(B)),
             and unique_unify(role(A), role(B)),
             and unique_unify(ref(A), ref(B)).

unique_unify(A,B) iff A is a variable or B is a variable
```

Figure 4.6
Unique Unification for Syntactic Chains

also notice that this kind of unification does not maintain the same semantics as standard unification. In standard unification a common instance of the two terms that unify is found, where common instance is defined as follows: (i) the common instance of two constants is the same constant; (ii) a common instance of a variable and a constant is the same constant; (iii) a common instance of two variables is a variable In our redefinition of unification, there is no admissible common instance of two constants. I do not explore the consequences of this modification here.

The mechanism with which chains are built in our parser is simple. A new chain is started, whenever a node which could be the head of a chain is found. Every subsequent empty element which is postulated by the parser must be able to unify with at least one of the chains.

4.4.3 The Computational Problems

In accordance with the linguistic facts presented above, several problems must be solved, when building chains. First of all, the parser has to decide whether to start a new chain or not. It also has to decide whether to start an $\overline{\mathrm{A}}$ or an A chain. Second, on encountering an empty element it has to decide to which chain it belongs. Third, it must check whether all the chains satisfy the well-formedness conditions. Finally, if not all the chains satisfy the well-formedness constraints, the parser can attempt to intersect or compose two or more chains in order to satisfy the well-formedness conditions. The first and second decision can be seen as instances of the same problem, which consists in identifying the *type* of link in the chain that a given input node can form (whether head, intermediate or foot). We can describe this sequence of decisions as four problems that must be solved in order to form chains: the Node Labelling Problem (NLAB), the Chain Selection Problem (CSEL), the Chain Intersection Problem (CHIN), and the Chain Composition Problem (CHC).

The Node Labelling Problem (NLAB)

The Node Labelling Problem can be informally described as follows.

(55) Given a node N to be inserted in a chain, determine its label L, where L $\in$ {$\overline{\text{A}}$H, AH, $\overline{\text{A}}$I, AI, $\overline{\text{A}}$F, AF, $\overline{\text{A}}$Op }.

This problem then describes a relation R: N $\times$ L, where N belongs to the set of nodes, and L belongs to the set of labels for the elements of chains. The algorithm to perform such computation is very simple.

Algorithm

Input: Node, Local Configuration
Output: List of Labels

1. If Node is lexical and [+wh] then Label $\leftarrow$ $\overline{\text{A}}$H.
2. If Node is lexical and [-wh] then Label $\leftarrow$ AH.
3. If Node is empty and has θ-role and has Case then Label $\leftarrow$ $\overline{\text{A}}$F.
4. If Node is empty and has θ-role and has no Case then Label $\leftarrow$ AF.
5. If Node is empty and has no θ-role and is in Spec of C then Label $\leftarrow$ $\overline{\text{A}}$Op or $\overline{\text{A}}$I.

6. If Node is empty and has no θ-role and is not in Spec of C then Label $\leftarrow$ AI.

The Chain Selection Problem (CSEL)

The Chain Selection Problem can be informally described as follows.

(56) Given a node N of label L, and an ordered list of chains C, return the chain C_i, possibly none, to which N must unify.

The algorithm to perform this computation is given below.

Algorithm

Input: Label(s), Ordered List of Chains
Output: Chain or empty set

1. if Label = $\overline{\text{A}}$H then start new chain
2. if Label = $\overline{\text{A}}$Op then start new chain
3. if Label = AH then start new chain
4. if Label = $\overline{\text{A}}$F then
   choose last element in ordered list of chains which is not saturated.
5. if Label = AF then choose (nearest) unsaturated chain.[3]
6. if Label = AI then choose (nearest) unsaturated chain.
7. if Label = $\overline{\text{A}}$I then
   choose last element in ordered list of chains which is not saturated, unless it is the immediately preceding element in the stack.

[3]Only one A chain at a time is possible (see previous section).

The complex condition in the last case is necessary to deal with subject-oriented parasitic gaps. In all other cases of structural indeterminacy between intermediate trace and empty operator, the feature unification mechanism will resolve, since the empty operator is postulated when all other chains are saturated. This could be thought as being derived from a rather reasonable axiom that prohibits empty CPs. A theorem that could be derived is that no "vacuous" Comp to Comp movement could be allowed. The ambiguity between $\overline{A}$I and $\overline{A}$Op is then resolved by looking at the previous token in the stack. If it is the head of an $\overline{A}$ chain, then a new chain is started. This strategy would handle sentences such as *A man [who [whenever I meet] looks old.]*

The Chain Intersection Problem (CHIN)

The Chain Intersection Problem can be informally described as follows.

(57) Given two chains, C_1 and C_2, return the chain C_3 (possibly none), which is the intersection of C_1 and C_2.

The conditions for the intersection of chains are given in (58).

(58) C_1 intersects C_2 iff
the head of C_1 c-commands the head of C_2
and C_1 and C_2 unify in syntactic features
and the head of C_2 is an empty category.

This problem is, therefore, simply the intersection of two sets under certain restricting conditions.[4]

The Chain Composition Problem (CHC)

The Chain Composition Problem can be informally described as follows.

[4] On inspection of the algorithms, one can notice that c-command is not checked. In the real implementation each node is given an identification number, which is then computed as an Huffman code once the tree is built. C-command can be computed as a function of the Huffman identification number of each node. The only restriction to the use of this technique is that the tree must be completely built, which is satisfied in this case. We assume that chains are built separately, and then intersected.

(59) Given two chains, C_1 and C_2, return the chain C_3 (possibly none), which is the composition of C_1 and C_2.

The conditions for the composition of chains are given in (60).

(60) C_1 composes with C_2 iff
the head of C_1 does not c-command the head of C_2
and C_1 and C_2 unify in syntactic features
and the head of C_2 is an empty category.

Notice that the only distinguishing condition between composition and intersection is the c-command requirement. In reality, chains that intersect, belong to the sets A and $\overline{A}$, respectively, while chains that compose are all $\overline{A}$ chains.

4.4.4 Discussion of the Algorithms

In Figure 4.7 I show schematically how these algorithms build chains. A pseudo-Prolog notation is used, which is similar to the output of the parser, where chains are represented as lists enclosed in square brackets. I show the I/O of the NLAB and CSEL algorithms, given the sentence *Who did you think that John seemed to like?*, where a multiple $\overline{A}$-chain and an A chain must be recovered. NLAB takes an input word and outputs a label, while CSEL takes a triple (Node, Label, Chains) as input, and returns a new chain list.

On Building Chains

This section discusses the problem of how features of chain links are determined by an efficient algorithm. A basic assumption of the theory presented in (Chomsky 1981; Chomsky 1982) about NPs, either lexical or nonlexical, is that they can be exhaustively partitioned by the features [±pronominal],[±anaphoric]. (See Appendix A for an argument to justify the existence of *pro* and its features [+pronominal,-anaphoric].) Chomsky (1982, 34) notices that trace and PRO are in (virtually) complementary distribution and that they (virtually) exhaustively cover the possible positions for NPs. Chomsky argues that this fact is explained if only one empty category is assumed, which is defined contextually. This is the so called *contextual* determination of empty categories: there

Who did you think e1 that John seemed e2 to like e3 ?		
NLAB	who	$\bar{A}$ Head
CSEL	who, $\bar{A}$ Head	[(who)]
NLAB	you	A Head
CSEL	you A Head	[[(who)][(you)]]
NLAB	e1	A Intermediate
CSEL	e1 $\bar{A}$ I [[(who)][(you)]]	[[(who,e1)][(you)]]
NLAB	John	A Head
CSEL	John A Head [[(who,e1)][(you)]]	[[(who,e1)][(you)(John)]]
NLAB	e2	A Foot
CSEL	e2 A Foot [[(who,e1)][(you)(John)]]	[[(who,e1)][(you)(John,e2)]]
NLAB	e3	$\bar{A}$ Foot
CSEL	e3 $\bar{A}$ Foot [[(who,e1)][(you)(John,e2)]]	[[(who,e1,e3))][(you)(John,e2)]]

Figure 4.7
Chain building example

is only one empty category that can take up different functions or occurrences in different contexts.

Brody (1984),Brody (1985) has shown that this argument is incorrect. Empirically, this interpretation would be supported by derivations where empty categories can change their status in the course of the derivation. The evidence for such derivations is unconvincing. (See Appendix A for a more detailed description of the arguments.) Conceptually, as Brody (1984) points out, the assumption that there exist only one type of empty category does not entail the existence of contextual definitions: a random characterisation, which is then filtered out by independently needed principles, would also work. Brody (1984) is devoted to showing that contextual definitions are totally redundant and can be eliminated from the theory without any loss of empirical adequacy and with gain for the explanatoriness and economy of the theory. Brody proposes to assign empty categories freely to one type or other, and let the principles of the grammar rule out incorrect type assignment. In this way the grammar is minimally redundant.

Brody's argumentation is very convincing, but what is a gain in explanatory adequacy for the grammar might not be so beneficial for the parser. Both Chomsky's and Brody's account of the distribution of nonpronominal empty categories rests on binding theory. The features assigned to empty categories are not an accidental collection, thus the account is explanatory. From the point of view of parsing this approach leads to inefficiency.

A strictly *principle-based* approach to the functional determination of empty categories, such as the one in Fong (1991), is bound to be inefficient, because not only are the properties of the empty categories determined by a random characterisation, as Brody suggests, which is filtered at a later stage, but it also rests on free coindexation. Fong (1990) has shown that the problem of enumerating all possible indexations is NP-hard. Even if some precomputation were allowed, for example by using Chomsky (1981) definitions directly, still the property of the empty category would depend on its binder. Since our initial problem is how to determine the binder of an empty category, the algorithm would have to be nondeterministic, guess a solution and then check its correctness. Such check would not occur until much later, when binding theory is tested, and unbuilding part of the tree might be required. As one more piece of evidence pointing towards the idea that functional determination of empty categories is not performed by the human parser, consider experimental results about parsing English and Japanese, performed by Fong and Berwick (1992), with a parser that uses functional determination of empty categories. They show that functional determination results in Japanese being parsed more slowly than English. This results is contradicted by human processing.

Correa (1988),Correa (1991) proposes a set of features for the functional classification of empty categories which is not based on the feature [±pronominal], [±anaphoric] and the relation to the binder, as proposed in Chomsky (1982), but rather on more local features, which do not require a potentially unbound search of the tree. The classification is shown in (61).

(61)

	A-position	Government	Case
wh-trace(foot)	+	+	+
wh-trace(intermediate)	-	+	-
NP-trace	+	+	-
PRO	+	-	-

Quite correctly, Correa points out that, by this classification, the type of a category can be determined without hypothesizing in advance to which chain the category belongs, since the conditions that determine its status are independent of chain formation.

In Correa's parsing rule, given a sentence structure, where the trace-antecedent relations are not identified, the interpretive chain rule computes coindexations. Each NP has an attribute *Chain*. Upon evaluation, each NP in a chain points to the next element in the chain (or to nothing if it is the last element, the foot of the chain). Moreover, nodes that can c-command an NP, such as IP, I′, CP, C′, etc. have two pointer-valued attributes: *A-Chain* and *AB-Chain*. The domain of evaluation of all these attributes is limited to a single production in the grammar. A single production defines, according to Correa, an "elementary tree", which is the local domain for feature assignment.

Empty categories are predicted structurally without computing their referent, *i.e.* without *identifying* them. In particular, we only need determine whether an empty category is properly governed or not, which we have seen can be done very locally by adopting Rizzi (1990)'s formulation and determine whether it should unify with the A chain or $\bar{A}$ chain. If one assumes that Case assignment is obligatory, then Case is the distinguishing feature of the foot of an $\bar{A}$ chain from the foot of an A chain.

On the other hand, the limit of one chain attribute per node, which is imposed by Correa's encoding, is incorrect. If the single rule is taken as the local domain, in the sense that all necessary feature assignment must be performed within its limits and no percolation of features is allowed, no chain could be built. Hence, this definition is too strict. Correa allows feature percolation. But he shows that an attribute grammar with unrestricted percolation of attributes corresponds to a type 0 grammar, thus it is too powerful. He imposes a single-slot restriction per type of chain, to limit feature percolation. By attaching only one AB-Chain slot to each node that could be on the connecting path between the trace and the antecedent, the attribution rule models some locality restrictions such as *wh*-islands and the Complex NP Constraint. These locality restrictions, though, depend on the language. This attribution rule would not work for less restrictive languages, such as the British variant of English (Grimshaw 1986) and Italian (Rizzi 1982), which allow multiple extraction, nor would it work parasitic gaps.

Restricting the Search Space

As the previous chapter on phrase structure has shown, computing features is not always profitable, as some features reduce the search space while others do not. To see that checking features does indeed pay off, the cost of checking these features must be compared to the benefit of reducing the search space.

This analysis mostly concerns the first algorithm, NLAB, which is constituted of a series of binary choices. More precisely, recall that the relevant information is a) whether a node is lexical or not; b) whether it has a theta-role or not; c) whether it has Case or not; d) whether it is a sister of C (hence it is in

$\bar{\text{A}}$ position) or not (thus counting as an A position). For the chain selection algorithm there are four main constraints: first, A nodes can only be inserted in A chains and $\bar{\text{A}}$ nodes can only be inserted in $\bar{\text{A}}$ chains. Second, empty nodes never start a new chain. Third, the closest head is always chosen as a potential chain to which unify. Finally, only unsaturated chains are chosen.

Consider what would result if NLAB did not check for all these factors. If b) is not checked, NLAB$'$ does not distinguish between feet and intermediate traces, even in the same type of chain, thus it outputs four sets of labels: AH, $\bar{\text{A}}$H, {AF, AI}, {$\bar{\text{A}}$F, $\bar{\text{A}}$I}. If c) is not checked, NLAB$''$ does not distinguish between A and $\bar{\text{A}}$ feet, thus it outputs AH, $\bar{\text{A}}$H, $\bar{\text{A}}$I, AI, {$\bar{\text{A}}$F, AF}. If d) is not checked, NLAB$'''$ outputs AH, $\bar{\text{A}}$H, {$\bar{\text{A}}$I, AI}, $\bar{\text{A}}$F, AF. If b), c) and d) together are not checked, NLAB$''''$ outputs AH, $\bar{\text{A}}$H, {$\bar{\text{A}}$I, AI, $\bar{\text{A}}$F, AF}.

In accounting for the growth rate in the space of hypotheses of these modified algorithms, two factors must be taken into consideration. One factor is the number of active chain types, namely whether a sentence presents only A or $\bar{\text{A}}$ chains, or both. This factor encodes the second and third restriction of the CSEL algorithm, with the consequence that not all combinations are attempted. The second factor accounts for the growth rate proper, which is reducible to counting the set of k-strings over an n-sized alphabet, hence n^k. Here k is the number of relevant links in the sentence (for instance, *feet* in NLAB$''$), and n is given by the size of the set of features collapsed by lifting some of these checks, hence 2,2,2 and 4, respectively.

The hypothesis space in the three algorithms grows in slightly different ways. In NLAB$'$, where there is no restriction on the number of active chains, the growth rate is n^k. For NLAB$''$ and NLAB$'''$ the formula is ${N_A}^k$, where N_A is the number of active chains. Practically, this amounts to 2^k at most, as the number of active chains is at most 2, because of the restriction on always selecting the nearest unsaturated chain. For NLAB$''''$ the restriction for active chains no longer holds, because in this algorithm no features are checked, so it is impossible to establish if a chain is saturated or not until structure building ends. Thus the growth factor is a function of the number of heads seen up to a certain point in the parse, the number of empty categories and also their respective order in the input. Notice that the different size of the collapsed feature set, which is larger for NLAB$''''$, is implicitly taken into account by k, as the number of relevant links varies with the size of the collapsed feature sets. For the same sentence there are more relevant links if the collapsed feature set is larger.

S	TL	NLAB′		NLAB″			NLAB‴			NLAB⁗	
		RL	G	RL	AC	G	RL	AC	G	RL	G
3	2	1	2	1	1	1	–	–	1	1	1
4	3	1	2	1	1	1	–	–	1	1	1
5	3	–	–	0	1	1	–	–	1	–	–
6	2	–	–	0	1	1	–	–	1	–	–
7	4	1	2	1	1	1	–	–	1	1	2
8	3	–	–	1	2	2	–	–	1	1	2
9	5	–	–	1	2	2	1	2	2	2	6
10	6	1	2	2	2	4	1	2	2	3	18

Table 4.1
Growth of Hypotheses Space: S = sentence; TL = Total number of links; RL= Relevant Links; AC = Number of Active Chains; G = Growth rate

Now, in all cases, the growth is exponential in the number of relevant links, while the possible gain obtained by not checking the features can be at most logarithmic in the number of potential empty categories. Since the number of potential empty categories is at most 2^f, for f binary features, this gain is expressed as f. Hence, suppressing feature checks becomes beneficial only if $kf > n^k$. Now notice that $2 \leq n \leq 2^f$. For $n = 2$ and $f = 3$, the inequality is satisfied for $k < 4$. This means that for algorithms NLAB″ and NLAB‴ all sentences with more than 3 relevant links are computed faster if features are checked. For $n = 4$, i.e. algorithm NLAB⁗ the inequality is never satisfied. [5]

The results of some calculations are reported in Table 4.1, which refer to the type of constructions exemplified in Figure 1.3 (sentence type 1 and 2 are not considered, because they only contain trivial chains.) If one considers a sentence such as *Who did you say that John thought that Mary seemed to like?*, with four gaps and four heads, there are 96 hypotheses about chain formation to explore using NLAB⁗. Clearly, checking features and using them for building chains, and keeping the hypotheses search space small, is beneficial in most cases.

Extensibility

These algorithms deal in detail with the somewhat neglected problem of what to do when more than one chain has to be constructed. They do not discuss

[5] Remark that here I am assuming that checking a feature and checking a chain have the same computational cost, which is an approximation, as a chain cannot be checked with a single operation.

specifically the issues of adjunction or rightward movement. However, they could be extended.

In the first place, the postulation and structural licensing of empty categories is always performed by the same mechanism. According to the ECP as formulated in Rizzi (1990, 25), Cinque (1990), and Chomsky (1986a), among others, for an empty category to be licensed, two conditions must be satisfied: the empty category must be within the maximal projection of a lexical head to be licensed structurally, and it must be identified by an antecedent. The structural licenser and the antecedent need not be the same element. In fact, they hardly ever are. Whether movement is to the left or to the right does not affect structural licensing (which is here performed by the conditions that apply to the reduction of an ϵ rule.)

Rightward movement requires an extension of the algorithm to incorporate the empty category in a chain. An empty category which is the foot of rightward movement must be licensed structurally, before its antecedent is seen. When the NP which is the antecedent (head of chain) is found, it starts a new chain, according to CSEL. Here an extension is needed to check if there are any empty categories waiting to be identified. This requires computing c-command, to check that the NP can be the antecedent of the empty category, if one is found. Explicit computation of c-command was not needed for leftward movement as it is a property of the stack of an LR parser that it encodes c-command. Thus the fact that a constituent contains an "orphan" empty category must be recorded, perhaps by composite categories. If the antecedent follows immediately on the stack the element that contains the empty category, c-command obtains (as a consequence of binary branching), and the empty category can be unified to the antecedent. [6]

4.5 PSYCHOLINGUISTIC SUPPORT

In general, the precomputation of syntactic features is supported by experimental evidence, and in particular, support is provided for using case information to distinguish the foot of an A chain from an $\overline{\text{A}}$ chain, as predicted by the ICMH.

[6] Alternatively, one could adopt the (linguistically radical) position that rightward movement does not exist (Kayne 1994). Although this generalization seems to be true for head movement, Kayne's proposal is more controversial for maximal projections. Discussion of these issues falls completely outside of the topic of the present work.

Plenty of work in the experimental psycholinguistic literature has addressed the question of how filler-gap dependencies are processed (Frazier, Clifton, and Randall 1983; Clifton, Frazier, and Connine 1984; Fodor 1989; Stowe 1986; Frazier and Clifton 1989). De Vincenzi (1991) presents psycholinguistic evidence in favour of a parsing strategy to build chains, called the Minimal Chain Principle (MCP) De Vincenzi (1991, 13))

(62) Avoid postulating unnecessary chain members at S-structure,
but do not delay required chain members.

This general parsing strategy is supported by several experiments. In particular, experiment 2 is designed to test the cost of the length of chains, where by length it is meant the number of links in a chain. Of course, if the computation of chains turned out to be insensitive to length, thus apparently a cost-free operation, much of the rationale for a parsing strategy as the MCP would vanish. De Vincenzi (1991) formulates the hypothesis tested in this experiment as follows.

(Experiment 2) The parser is sensitive to syntactic features such as Case, and it uses them in building chains. Building chains is a costly operation.

Superficially minimal pairs that vary only in length with the input can be constructed in Italian by using contrasts between unaccusative and unergative verbs. (See Appendix A for definitions and examples.) In short, since the inverted subject of an unaccusative verb is base generated, while the inverted subject of an unergative is moved, sentence (63)b should take longer than (63)a, according to the MCP.

(63) a. E' arrivato un bimbo.
"A child arrived."
b. Ha esitato un bimbo.
"A child hesitated."

For the result to follow, however, one more crucial assumption about Case for unaccusatives must be made. Namely, that the parser is sensitive to Case assignment at this stage of structure building. In fact, according to Belletti (1988), unaccusative verbs can assign inherent partitive case. Since partitive Case, though, is incompatible with a definite interpretation, a postverbal definite NP is adjoined to the VP, and it is not in its base generated postverbal position, otherwise it would not receive Case. Thus, structurally there is a difference between (64)a and (64)b.

(64) a. E' $[_{VP}$ arrivato un bimbo.]
"A child arrived.
b. E' $[_{VP}$ arrivato $[_{VP}$il bimbo.]]
"The child arrived."

If the parser did not use information about Case at all at this stage, (but it used it only in the discourse, see discussion in De Vincenzi (1991, 51ff)), then it would assume that (64)a does not receive Case, and it would adjoin it to VP, similarly to (64)b. In this case we would predict longer times in (64)a than in (64)b. The experimental results confirm the MCP and the use of Case.

The fact that the parser uses Case is an implicit consequence of the MCP, as shown by experiments 1 and 5 conjunctively. Experiment 1 confirms the following hypothesis: the parser obeys the first clause of the MCP by always trying to interpret elements in their base generated position, *i.e.* the parser always tries to build singleton chains. Experiment 5 support this other hypothesis: the parser obeys the MCP and the principles of the grammar, *i.e.* the MCP is not used if it causes a violation of some principle of the grammar, specifically the relevant one, namely the ECP. If the parser honours the principles of the grammar (Experiment 5), then when interpreting an NP as a singleton chain (Experiment 1) it will immediately check that the chain receives Case and θ-role, and therefore is well-formed. This result is also consistent with use of morphologically realized case to disambiguate locally ambiguous input.

This piece of evidence corroborates the proposed classification of syntactic features, and especially the use of Case to distinguish the foot of an A chain from the foot of an $\overline{A}$ chain. The use of syntactic features is crucial for the structural determination of empty categories which we adopt. By using structural determination, empty categories can be licensed locally, within their maximal projection.

4.6 INCREMENTAL ASSIGNMENT OF FEATURES

On inspection of the presented algorithms for chain formation and the examples of input shown in Figure 4.7, one can notice that Case is assigned immediately upon reducing the NP, thus enabling the chain building algorithms to distinguish between different types of chains. In particular, case features must be assigned to an empty element of the chain immediately, as case is crucially used by NLAB, to determine if the empty category belongs to an A or an $\bar{A}$ chain. In general, feature assignment is performed from left to right, while scanning the input. One need, therefore, tackle the issue of incrementality in LR parsing.

LR parsers have been criticized as possible models of linguistic performance because they build mother nodes only after all children are seen, and therefore they have worst-case behaviour on right-branching structures (Abney 1989; Steedman 1989).

Evidence for the Incremental Parsing of Correct Input

The most impressive evidence that human parsing is incremental is provided by Marslen-Wilson's experiments on shadowing Marslen-Wilson (1973). Fast shadowers are able to restore errors while they shadow speech at one-syllable distance. This fact is interpreted as evidence that they process speech and that they perform lexical access as soon as the input signal is received.

Surprise effects in sentence parsing in head final languages provide evidence that syntactic parsing is performed incrementally even when information is insufficient. In the Japanese sentences in example (65), speakers report a surprise effect at the word *tabeta*, which means that they had already started processing the three arguments, possibly as belonging to a yet unseen predicate.

(65) Bob-ga Mary-ni [$t_{nom/i}$ ringo-wo tabeta] inu-wo ageta
Bob-NOM Mary-DAt apple-ACC eat-PAST dog-ACC give-PAST
'Bob gave Mary the dog which ate the apple'

These two examples show that people integrate incoming material very fast, and that they start interpreting it before the end of the sentence is reached. It appears reasonable, for the purpose of investigating syntactic parsing, to assume that the input is processed word by word.

Different researchers have interpreted results on incrementality as stemming from different properties of the processing system. Some researchers require that each incoming word be attached to one single connected structure (Crocker 1992; Sturt and Crocker 1995; Stevenson 1994; Stabler 1994; Wehrli 1992).

The basic argument for building as fully a connected structure as possible are based on memory requirements: we know that storing disconnected items is more costly than storing whole units. In practice, some words have to be pushed temporarily on a stack. Stabler (1994) argues in favour of assuming a fixed, small bound for the number of items that can be left on a stack. [7] Other researchers extend the observation on fast integration of lexical material to an overarching requirement to maximize intepretation, which would justify the parser in projecting unseen structure, when it can be safely predicted (Crocker 1992; Crocker 1995). Finally, one should notice that there are advocates of incremental parsing for engineering goals. Tomita (1986),Wehrli (1992) report on interactive parsing systems, which require incremental parsing.

Several counterarguments to the supposed lack of incrementality of LR parsing have been constructed. two assumptions underlie the argument that LR parsing cannot be incremental: the first assumption is that interpretation cannot start before the completion of a rule in the parser, i.e. that the syntactic chunking units must correspond to the interpretation chunking units, a sort of simplicity arguments; the second assumption is that the crucial source of information about the history of the parse is the stack. Stabler (1991),Shieber and Johnson (1993) show that both these assumptions are false. Stabler (1991) argues that interpretation can start before the end of the right handside of a rule is reached if a sort of postfix notation is used. This observation is in the right direction, because it points out that the simplicity argument used by Steedman is incorrect. Steedman assumes that interpretation corresponds to building a constituent and integrating it into the syntactic structure, and that the best parser is the one that does this most "simply", namely with a minimal manipulation of the grammatical representations. Incremental integration and interpretation of constituents in a bottom-up parser can occur only if the structure is left-branching. Therefore he argues, syntactic theory should build left-branching representations, such as those of categorial grammar. This argument rests on the assumption that syntactic interpretation can occur only when one constituent has been built; in LR parsing terms, when a rule has been reduced. This assumption requires synchronization of the parser with

[7] Only fully top-down models could build a structure that is fully connected at every step. Top-down parsers are unattractive, because they are liable to postulate too much structure, incurring "spurious" ambiguities. Moreover, they do not terminate on left-recursive input.

the interpreter. In fact, asynchronous computation is actually simpler than synchronous computation Shieber and Johnson (1993).

However, to rebutt the critique completely, one must show that LR parsing can build a connected structure. Drawing on work by Lang (1974),Lang (1989), Shieber and Johnson (1993) show that the states of an LR parser contain enough information to build the structure incrementally. An example of this technique is provided in Figure 4.12 in the following section.

Since incrementality provides evidence in favour of top-down-like behaviour, it is important to remember that evidence for bottom-up behaviour comes from observations on the speed at which language is parsed. Also, universal properties of human languages, as Hawkins (1990) argues, are more easily interpreted, if they follow from a requirement to parse languages bottom-up.

Hawkins (1990) argues for a processing explanation for word orders, as one finds left-right asymmetries in the world languages, with a clear preference for structures in which an immediate-constituents-to-words ratio is minimized. A processing explanation of word orders is based on the assumption that the parser projects all the structure it can as soon as it has unambiguous evidence for it (but not before). So for example, if C_0 is an unambiguous index of a CP, the whole CP structure will be projected. If parsing did not proceed bottom-up, it would make no sense to try and minimize the processing window. Analogously, one can argue that a parser that is fully predictive, i.e. completely top-down, would have no reason to prefer lower immediate-constituent–to-words ratios.

In general, as Stabler (1994) points out, there seems to be contradictory demands on the parser, to build connected structure to minimize the use of memory resources, requirement which is maximallysatisfied by top-down techniques, and to build structure only if supported by input evidence, for efficiency and descriptive reasons, requirement which is maximally satisfied by bottom-up algorithms.

Evidence for Incremental Parsing of Incorrect Input

Few parsing methods can model the available evidence on incorrect input. Briefly put, there is evidence from mispronounciation detection experiments that humans detect an error in the input as soon as they hear it (Cole and Jakimik 1978).

Cole and Jakimik (1978) employed the technique of mispronounciation detection to show that speech is processes word by word. They presented pairs of words, opposing pairs in which the second word was highly constrained by the preceding word (*gold ring*) to pairs in which the second word was not (*old ring*). Subjects pushed a response key whenever they detected a mispronounciation. Reaction times were 180 milliseconds faster for predictable words, than for non-predictable words. This shows that previous context influences word processing, and that speech is processed word by word, as the previous word must have been processed to be able to affect the following word. The average detection time was around 800 milliseconds, which means that averagely the mispronounciaton was detected very fast, as it was recognised by the end of the following word.

Interestingly, early detection of incorrect input is a very discriminating feature among parsing models. Most parsing models allow the parser to shift a token on some temporary stack, if no other attachment or action can be performed, thus often the parser proceeds until the end of the input, or until some threshold is reached before realising that the parse is incorrect. Since all sentences that contain an error are incorrect, there is no notion of "local" error, parallel to the notion of local ambiguity. Thus, only a sufficiently predictive parser could detect the error and suspend the parse right away, or trigger some recovery procedure.

In the ligth of the evidence just reviewed, I propose a comprehensive definition of incrementality, that takes into account the properties of the parser both on correct and incorrect input.

Definition

Incremental Parsing

A parser is incremental if it has the following properties:

1. It has the valid prefix property, namely it does not perform any unnecessary shifts on incorrect input, because it is able to recognise incorrect input as soon as possible,
2. At each point in the parse it builds the smallest possible forest.

Moreover, I want to impose a requirement that the proposed algorithms be valid cross-linguistically.

Definition

Cross-linguistic Incrementality

A parser is incremental for all languages, if it uses the same incremental parsing mechanisms for all languages.

I shall argue that only precompilation of the grammar, with an exhaustive encoding of all the possible acceptable parses, can make the parser sufficiently predictive, and thus incremental on both correct and incorrect input.

Other approaches

While most parsers satisfy property 2 above (including LR parsing, as argued by Shieber and Johnson (1993), and as I will show below in more detail), very few parsing methods have property 1, and many can be shown not to be valid cross-linguistically.

Many results on left corner parsing appear to show that it is more apt to describing evidence on human sentence parsing, especially processing overload (Abney and Johnson 1991; Gibson 1991), but if one looks at a well-defined proposal, one can reach the conclusion that the incrementality of the algorithm depends on the head-initial nature of English, coupled with its rather fixed word order.

Let's examine in detail the proposal reported in Stabler (1994), which is precise and well-defined. Stabler presents a variation on left-corner parsing, called Left Attachment, which has the property of parsing a sentence without ever pushing more than two elements on the parse stack, thus it maintains what he calls *finite connectivity*. Finite connectivity is a relaxed version of incrementality, which, instead of imposing that the parser always build a connected graph, puts a limit on the number of elements that can be left unconnected. Incrementality is a sort of finite connectivity where the limit of unconnected elements is set to 1.

In Stabler's proposal, finite connectivity is achieved by two means: rule composition, and the use of a variable trigger function. The first line of Figure 4.8 shows an example of rule composition. The NP can be shifted on the stack, because it is recognized bottom-up by the sequence of rules NP $\rightarrow$ N and N $\rightarrow$ the – farmer. Using a variable trigger function means that instead of always using the left corner of the rule to trigger top down prediction, the portion of the righthand side of the rule that triggers recognition can be of variable length. In particular, in this proposal, the specifier of a maximal projection is projected bottom-up up to X′, while the XP and the righthand side sister, down to the X′, are predicted. This algorithm is particularly effective because English is head-initial, so that the head is actually the left corner of the subtree rooted in X′. In this way the parser can predict the XP based on left corner information. Because the head is on the left of the X′ subtree, it can be immediately projected to X′ and be attached to the predicted X′ node. The parse would be less effective for head-final languages.A language where the head is final could not exploit rule composition, as the complements would come first, and complements are not nearly as predictive of the type of projection being processed as heads are. The variable trigger function would not be very effective for very similar reasons.

In order to see that this is actually so. consider Figure 4.8 which traces a parse of the Japanes sentence in (66), a head final language, with phrase structure that is more or less the mirror image of the structure of the English examples in Stabler (1994).

(66) Noumin-ga kitune-wo oikaketa
farmer-NOM fox-ACC chased

This parse uses Stabler's predicates, with adaptation for a head final language. The notation *XPshift: NP* $\Rightarrow$ *the-fox chased* means that as a consequence of the application of the XPshift rule there is an NP on the stack and *the-fox chased* is the yet unparsed input. It is easy to see that even if we make adjustments to the algorithm to optimize for head final languages (for instance, by using an *X1shift/attach rule*), the parser could stack all the complements in a left-branching structure.

I follow here the notation proposed by Stabler, and of course his axioms and parsing rules. In the sentence in Figure 4.8, for instance, the parser starts from the parsing axiom: the empty stack (at the left of the double arrow) and the input *the-farmer the-fox chased*. From this configuration and the combination

1	ax: ⇒ the-farmer the-fox chased	⇒ NP → N	⇒ N → the-farmer
2	XPshift: NP ⇒ the-fox chased	⇒ IP → NP I1	
3	lc: $\overline{I1}$ IP ⇒ the-fox chased	⇒ NP → N	⇒ N → the-fox
4	XPshift: NP $\overline{I1}$ IP ⇒ chased	⇒ VP → NP V	
5	lc: $\overline{V}$ VP $\overline{I1}$ IP ⇒ chased	⇒ V → chased	
6	shiftX0/attach: VP $\overline{I1}$ IP ⇒	⇒ I1 → VP I0	⇒ I0 → empty
7	shiftX1/attach: IP ⇒		

Figure 4.8
Parsing Japanese with the Left Attachment Algorithm

of rules shown on the first line, the parser can deduce that it can apply an XPshift action, where an NP is recognized and shifted onto the stack (second line). From this configuration, in conjuction with the grammar rule IP → NPI1 the parser can predict an I1 (the overline means that the nonterminal has been predicted top-down and needs to be confirmed by the incoming input bottom up). The result of this operation, the application of the left corner (lc) rule, is shown on the third line. Line 4 shows the result of recognizing the second NP, by the same combination of rules shown before. The NP is shifted on the stack. Line 5 shows that having I1 and NP on the stack triggers a left corner prediction of the upcoming verb (which in japanese is in final position). When the verb is actually consumed (line 6) it can be immediately attached without being actually shifted on the stack using an operation that combines predictable sequences of actions and keeps the stack small. Line 7 shows that the same predictable sequence of actions can be done at the X1 level, and the sentence is recognized. It should be noted that even in such a small example the limit of 2 tokens on the stack is already too strict. In this particular parse, the limit of finite connectivity is overflowed in the deduction step indicated from line 3 to line 4, when the NP *the-fox* is recognized. The NP cannot be attached to the I1 directly, because the I1 is only predicted, but it awaits bottom up confirmation. Thus, it is necessary to explore more powerful parsing techniques, as I illustrate below.

4.6.1 Incremental LR Parsing

Property 1 is a well-known property of LR parsers, and similarly compiled algorithms, for instance Schabes (1991). Properties 2 and cross-linguistic validity can be achieved if the LR algorithm is augmented with two standard techniques: the use of marker non-terminals, and the interpretation of the LR states (following the idea in Shieber and Johnson (1993).) I will extend Shieber and Johnson (1993)'s argument to head-final languages, by showing examples from Japanese. Differently from them, I will concentrate on showing that feature assignment can be performed incrementally, and I will not discuss interpretation. I consider these as instances of the same parsing problem.

S-attributed and L-attributed Grammars

In order to perform feature assignment incrementally, an LR parser must be able to assign features at any point in a rule. Compare, for instance, (67)a and (67)b.

(67) a. IP $\rightarrow$ NP I VP { assign Case to NP, if I= finite }
b. IP $\rightarrow$ NP I { assign Case to NP, if I= finite } VP

In a parser that uses rule (67)a, case assignment to the Spec of IP position, *i.e.* to the subject, is performed only after the VP is seen, even if no information about the VP is needed to perform the case-assigning action. This means that, if IP is the root, Case assignment to the subject is going to occur only when the entire tree for the sentence is built. A parser that uses rule (67)b, on the other hand, would assign Case as soon as the necessary information is available. Although it is *literally* true that LR parsers can evaluate actions only on reductions, *i.e.* they only operate on grammars whose rules have the form in (67)a, there are simple techniques to transform a rule like (67)b into a set of rules like (67)a (Aho and Ullman 1977, 282ff).

Stating the problem more precisely, A grammar G with productions of the sort shown in (67), *i.e.* a grammar that can perform attribute assignment upon reduction of a rule is called an *attribute grammar*.[8] An *attribute grammar* is *S-attributed* if all the attribution rules have the form in (68).

[8] It was first proposed by Irons (1961) and formalized by Knuth (1968). For a clear example of how to use attribute grammar for natural language, see Correa (1988),Correa (1991).

(68) $A.a \rightarrow B.b\ C.c \qquad \{ A.a \leftarrow f(B.b, C.c) \}$

The rule in (68) is called S-attributed because the attribute of the parent node is a function of the attributes of the offspring. Such attributes are called *synthesized* attributes. An *attribute grammar* is *L-attributed* if all the attribution rules have the form in (69).

(69) a. $A.a \rightarrow \beta.b \qquad \{ \beta.b \leftarrow f(A.a) \}$
b. $A.a \rightarrow \beta.b\ \gamma.c \qquad \{ \gamma.c \leftarrow f(\beta.b) \}$

The rules in (69) are called L-attributed because the attribute of a vocabulary symbol is a function of the attributes of a preceding item in the rule, or of the parent node. Such attributes are called *inherited* attributes. An L-attributed grammar G_L can be evaluated by an LR parser, if G_L is transformed into a grammar G_S such that the actions that perform attribution in G_L always occur at the end of a production in G_S.

This transformation can be achieved by two different means, depending on the situation. One can use the marker non-terminal technique, which simply consists in adding a "dummy" nonterminal, in order to force rule reduction at a particular point. This technique can solve the problem of interleaving feature assignment and structure building in two interesting cases. By putting a marker non-terminal right after the verb, Case is assigned to the NP as soon as possible in order to distinguish A from $\bar{A}$ chains, and enable structural determination of empty categories. If Case morphology is overt, the marker non-terminal can be used to assign Case and θ-role, so that participants ina predicate where the verb is final can be interpreted.

The marker nonterminal technique cannot be used if the grammar has rules where the attributes percolate from right to left, for example in a rule like (70).

(70) $A \rightarrow B\ C \qquad \{ B.i \leftarrow C.s \}$

For natural languages this technique does not work for SVO languages to assign features from the verb to the subject, and for SOV languages to assign the features of the verb to the arguments.

In the former case, the transformation of an L-attributed grammar into an S-attributed grammar, can be performed by different means, because in many cases of L-attribution, the attributes of the tokens on the left are at a fixed position in the stack. For instance, consider the following example, which is an adaptation of Aho and Ullman (1977, 310), where they give an example for programming languages. Let a grammar G, with productions P shown in (71) be given.

(71)
1. DP $\rightarrow$ Det Poss NP
2. NP $\rightarrow$ NP e N
3. NP $\rightarrow$ N
4. N $\rightarrow$ { *cani, gatti, pulcini, ...* }
5. Det $\rightarrow$ { *i, il, la, ...* }
6. Poss $\rightarrow$ { *miei, mio, mia, ...* }

Let the input string in (72) be also given.

(72) I miei cani e gatti e pulcini
My dogs and cats and chickens

We assume that agreement is checked on applying the rule (71).1. The sequence of moves made by an LR parser is shown in Figure 4.9.

On inspection of the parse trace, one can notice that every time *NP* is reduced, *Poss* is the element just below it in the stack. This means that the position of the element with which the list must agree is known. Case assignment into the Spec of IP (nominative Case or structural case) can be done in the same way, as in English structural Case can be assigned as a function of the next token, or a bounded amount of stack, or both. In English, structural case is assigned to the Spec of IP position, by Spec-Head Agreement with a finite Infl.[9] Spec of finite IP can occur both in main and embedded clauses. Moreover, we assume, that the finite morpheme is always in Infl, either because it is base generated there, or, as for auxiliaries, it moves to I (Pollock 1989), in English. Finally,

[9]We are not interested in structural case assignment to object position, *i.e.* inherent or partitive Case (Chomsky 1986a; Belletti 1988), as they involve Case assignment to a position after the head, in English, which can be treated by ordinary S-attribution.

STACK	INPUT	PRODUCTION
-	i miei cani e gatti e pulcini	
i	miei cani e gatti e pulcini	
Det	miei cani e gatti e pulcini	Det $\rightarrow$ *i*
Det *miei*	cani e gatti e pulcini	
Det Poss	cani e gatti e pulcini	Poss $\rightarrow$ *miei*
Det Poss *cani*	e gatti e pulcini	
Det Poss N	e gatti e pulcini	N $\rightarrow$ *cani*
Det Poss NP	e gatti e pulcini	
Det Poss NP *e*	gatti e pulcini	
Det Poss NP *e gatti*	e pulcini	
Det Poss NP *e* N	e pulcini	N $\rightarrow$ *gatti*
Det Poss NP	e pulcini	NP $\rightarrow$ NP *e* N
Det Poss NP *e*	pulcini	
Det Poss NP *e pulcini*		
Det Poss NP *e* N		N $\rightarrow$ *pulcini*
Det Poss NP		NP $\rightarrow$ NP *e* N
DP		DP $\rightarrow$ Det Poss NP

Figure 4.9
Example of L-attributed Incremental Parsing

English is head-initial. As a consequence of the interaction of these properties, Infl is always the next token in the lookahead of the parser, when the reduction of the NP which is going to occupy the Spec of IP is performed.[10]

In other words, structural case is assigned in the rule (73).

(73) $\mathrm{IP} \rightarrow \mathrm{NP}$ {Case assign, if Infl = +finite } I'

This rule assigns case correctly, only if the attribution is not a function of the other elements in I'. We notice however, that this is precisely what is meant by *structural* case: a case that is assigned independently of the properties of the main verb, in a given configuration. Thus, the correctness of (73) is a consequence of (74).

(74) a. $\mathrm{X0} \rightarrow \mathrm{w0}$ $\{ \mathrm{X0.Case} \leftarrow \mathrm{w0.Case} \}$
b. $\mathrm{X1} \rightarrow \mathrm{X0}\ \mathrm{Y2}$ $\{ \mathrm{X1.Case} \leftarrow \mathrm{X0.Case} \}$
c. $\mathrm{X1} \rightarrow \mathrm{X0}$ $\{ \mathrm{X1.Case} \leftarrow \mathrm{X0.Case} \}$

The rules in (74) show that the rules of attribution that transmit case from the inflectional input token (by which we mean modals, auxiliaries, and finite

[10] At first sight, this might appear as a wild overidealization. In fact, there are both theoretical and empirical reasons to think that this is the right way to idealize the data. A corpus analysis on 111 occurrences of the verb *announce* in the Penn Treebank shows that the subject is followed by an aspectual adverb 11 times, twice by incidental phrases, and 4 times by an apposition. In all other cases the subject and the verb are indeed adjacent. I do not consider appositions and incidentals as challenging for the general claim: incidentals are clearly outside of an $\overline{\mathrm{X}}$ structure assigned to the sentence; while appositions are "internal" to the NP, thus when the verb is reached, the phrase sitting on the stack is indeed the NP subject. which can therefore receive Case. The treatment of aspectual adverbs is more complex. There are at least two possible tacks. First, one can notice that adverbs, although they are analysed as maximal projections because they can be modified, never take a complement, thus they are usually limited to a very short sequence of words, and they do not have a recursive structure. A minimum amount of lookahead, even limited to these particular instances of aspectual adverbs, would solve the problem. Clearly, this is an inelegant solution. A more principled treatment comes from recent developments in the theory, that have changed somewhat the representation used for adverbs. Laenzlinger (1993) suggests that all maximal projections have two specifiers, one A and one $\overline{\mathrm{A}}$, the higher of the two is the $\overline{\mathrm{A}}$ position which can be occupied by adverbs, if they are licensed by the appropriate head (the Adv-Criterion). For these adverbs the appropriate head is Asp_0 which we find only with finite verbs. The parser could compile this information and assign case directly, without even waiting to see the (lexical) verb.

verbs), here represented by w0, are *copy rules*, hence they simply transmit a lexical property from the input token to the sister of the assigner X1. Hence Case can be assigned to the NP directly.

Extension of the Attribution Schema

This technique, which is sufficiently flexible to assign Case to the subject, extends to other languages with properties different from English, but it seems to be restrictive enough to provide an explanation of a gap in the typology of languages.

In verb-final languages like German, the Spec of IP is not string adjacent to the head of IP, like in English, in embedded clauses. Structural case is assigned from left to right, since the complementizer, which necessarily marks the left edge of an IP, is obligatory, and the finite complementizer is always different from the infinitival complementizer, as shown in (75).

(75) a. Wir wissen, daß Marie das Buch gelesen hat.
"We know that Mary has read the book."

b. Marie ist in die Bibliothek gegangen, um das Buch zu lesen.
"Mary has gone to the library to read a book."

A language like Japanese, which is head-final and has an optional complementizer to mark embedded sentences, which comes at the end, and has no relative marker for relative clauses, seems, at first sight, to constitute a problem. But in fact it does not, for two reasons: first, Japanese has overt case marking, hence nominative case assignment can be detected as it is explicitly marked in the input; secondly, Japanese is left branching, hence the grammar to describe Japanese could be evaluated by using exclusively S-attribution.[11]

An LR parser could not assign features incrementally in a language with the characteristics in (76).

[11]Weinberg (1993) presents a similar idea for a parameterized deterministic procedure to parse both English and Japanese. Her main idea is that internal licensing within a phrase is done by the head in English, and by Case marking in Japanese. She shows that garden paths effects follow.

(76)
1. no overt case marking
2. no distinct finite complementizer
3. verb final
4. right branching

A first inspection of some of the sources on language typology show that such language might be very difficult to find. (Steele 1978; Shopen 1985; Comrie 1981). According to Downing (1978), verb final languages usually have prenominal relative clauses, which we take to be a sign that they are left branching. Only two verb-final languages have postnominal relative clauses, Persian and Turkish. In Persian the clause boundary is overtly marked by the suffix *-i* on the antecedent. Moreover, they both have overt case marking of the subject. In fact, Downing (1978), noticing this rather strict implicational universal, attempts an explanation in terms of parsing theory by citing Kuno (1974). He says:"Kuno (1974) has shown that SOV word order with postnominal relative clauses maximizes center-embedding. which seriously interferes with sentence processing beyond two degrees of embedding. It is not surprising then to find that languages with post rather than prenominal relatives provide correlative structure as an alternative." (Correlative relative clauses are those that have the syntactic structure of main clauses.) Although this is by no means definite evidence, at least it suggests that the algorithms for chain formation and feature assignment that we have presented are not immediately falsified and they are applicable to a wide variety of languages with different properties.

Interpretation of the LR states for head-final languages

Even if the annotation techniques presented in the previous sections seem to cover an interesting range of syntactic facts and gaps, one might want to explore a more powerful technique.

The feature assignments explored above all fall short of one requirement for incremental parsing: they do not build structure incrementally. In fact, they do not build structure at all, the reason being that they operate only on the stack. The stack of an LR parser encodes some interesting linguistic concepts, such as precedence and c-command explicitly. It encodes structure only implicitly, as Shieber and Johnson (1993) remark, as it encodes the states the parser has gone through. However, the complete structural information is there, and can be used to parse incrementally both head-initial and head-final languages.

S′	→ S	I0	→ empty
S	→ CP	VP	→ v NP
S	→ IP	VP	→ v
CP	→ NPwh C1	VP	→ v IP
C1	→ C0 IP	VP	→ VP PP
C0	→ do	NP	→ n
C0	→ empty	NP	→ det n
IP	→ NP I1	NP	→ NP PP
I1	→ I0 VP	PP	→ p NP

Figure 4.10
English grammar

Consider a grammar with the set of productions P, shown in Figure 4.10. Its canonical collection of items, from which the LR finite state machine is derived, has 28 states. I show the first 6, to illustrate the technique, in Figure 4.11.

These states encode top-down information implicitly. For example, the first item in I_5 constructs an NP as the first daughter of an IP, as the parent of the node is known by consulting the very same state. The parent of the IP is also known, as it must be in state I_0, the state preceding I_5 in a graph traversal. The parent of IP in this case is the root of the sentence. That the whole history of the derivation of a constructed item in an LR graph can be computed becomes perhaps clearer if one thinks of all the shift operations as the traversal of a finite state graph, the graph of all the left contexts.

This information could be used as soon as the parser enters into a given state. Whenever the parser enters in a state, it predicts all the structure that is right after the • in a rule. This information can be encoded if we think of a tree structure as an AND/OR graph, where nodes that belong to the same derivation are AND nodes, while nodes that belong to different derivations are OR nodes (Lang 1989; Shieber and Johnson 1993). Visually, the nodes that belong to the same tree are represented as ordinary nodes, while nodes that represent disjunction are represented as ovals. Figure 4.12 shows a representation of the state of the parser at states 7,5, and 13, for the input *the girl sent*.

In the first panel, the parser has seen the determiner *the*, which is sufficient to predict that the upcoming phrase is either an NP or an IP, since the nonterminal Det can only be the first symbol of an NP which in turn is the first symbol of an

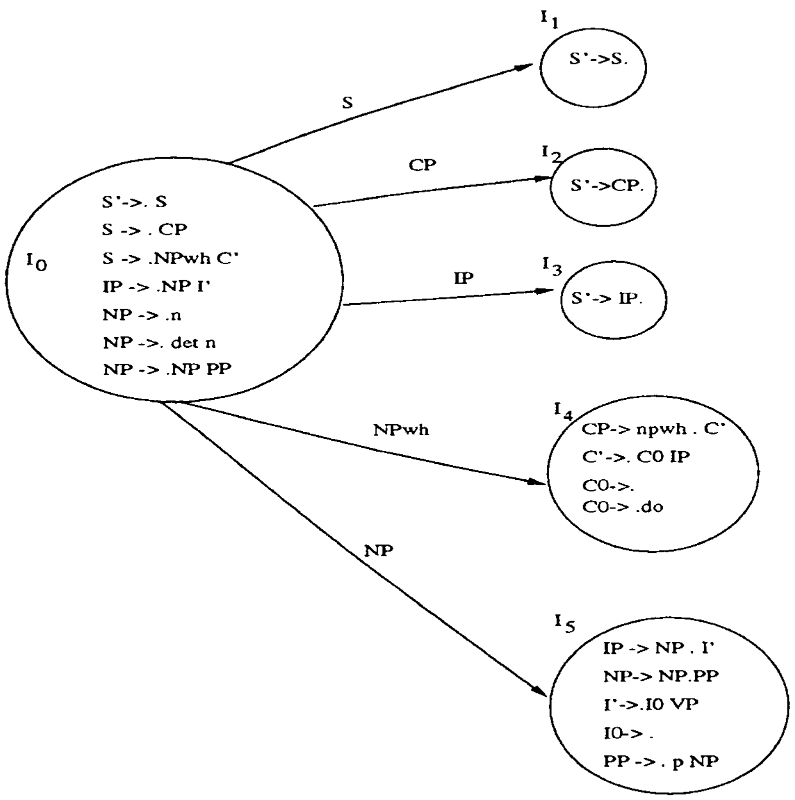


Figure 4.11
Canonical Collection of Items for the Grammar in Figure 4.10

IP. The shaded areas indicate the portion of the tree which has been confirmed bottom-up. The dotted branch in the figure in the NP projection indicates only dominance and not immediate dominance, as the grammar employs a left recursive rule at this point, so more than one NP could be constructed. The oval surrounding the two roots indicates that this is an OR node in the graph, thus the two trees rooted here are competitors analyses for the same input. One of the two will be discarded by the end of the sentence. The second panel shows the parser's state after the word *flowers*, which has been recognized as an NP, and attached to both subtrees in the AND/OR graph. The NP continuation is compatible with both analyses, as shown by state I_5 in Figure 4.11. The third panel shows that the NP analysis has been discarded at the processing of the verb *sent*, as it is incompatible with a V expansion. On the other hand, the tree rooted in IP has now an OR node which contains three possible expansions, all of which are compatible with the portion of input seen so far.

This use of LR states is very powerful, as it amounts to bottom-up parallel parsing. The extension to head-final languages is straightforward. Consider, for instance, the little, purely illustrative grammar for a head-final language given in Figure 4.13. This grammar could generate, for instance, the sentence in (77). The parse of this sentence would proceed as in Figure 4.14, where the first steps are illustrated.

(77) Bob-ga Mary-ni inu-wo ageta
Bob-NOM Mary-DAt dog-ACC give-PAST
'Bob gave Mary the dog'

In step 1, the main sentential structure is constructed, on the basis of the information contained in the first NP. The first NP being nominative, it can only be the left corner of a sentential node. Step 2 shows that, as soon as an inflectional node is postulated, all the possible VP expansions are predicted, as alternative analyses, indicated by the OR node. Step 3 shows that only one analysis is kept on encountering the dative NP, which is not compatible with the first and third expansion of the VP indicated in step 2. The parse will continue successfully, and unambiguously, with the analysis of the NP and of the verb.

This technique is powerful enough to parse head-initial and head-final languages incrementally, without losing the valid prefix property, and thus it satisfies the requirements we have imposed above. Since it is based on the collection of items, which take the closure of dotted items, it is also guaranteed to terminate

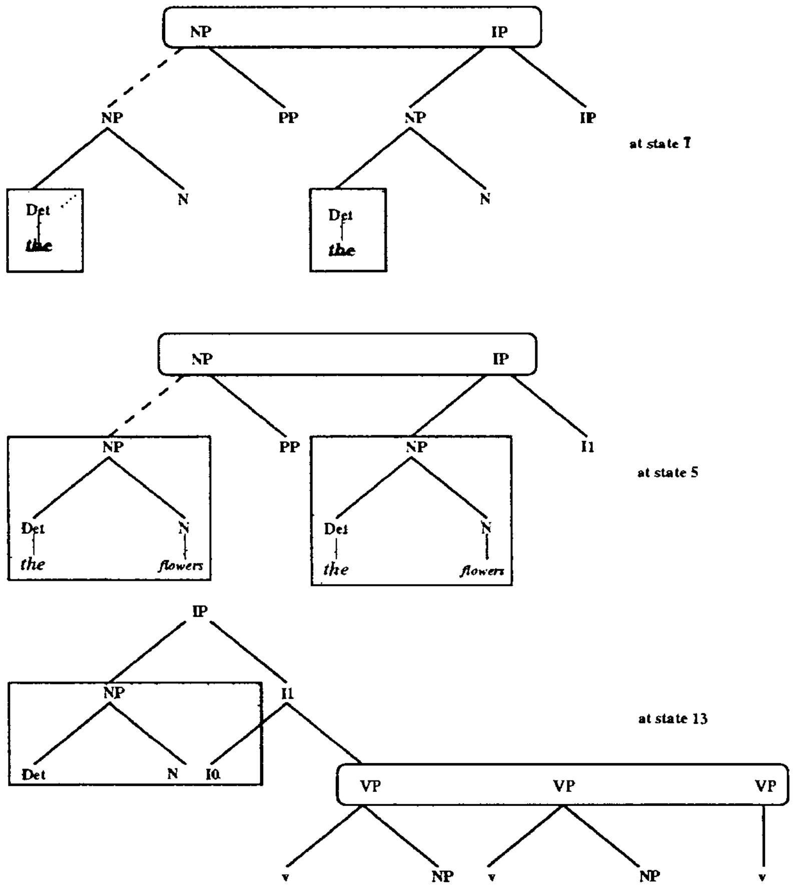


Figure 4.12
The three initial parsing step for the sentence fragment *The girl sent..* using a representation of the sentence as an AND/OR graph

$$\begin{array}{ll}
S' & \rightarrow \text{IP} \\
\text{IP} & \rightarrow \text{NPnom I1} \\
\text{I1} & \rightarrow \text{VP I0} \\
\text{I0} & \rightarrow \text{empty} \\
\text{VP} & \rightarrow \text{NPacc v} \\
\text{VP} & \rightarrow \text{NPdat NPacc v} \\
\text{VP} & \rightarrow \text{IP v}
\end{array}$$

Figure 4.13
Example Verb Final Grammar

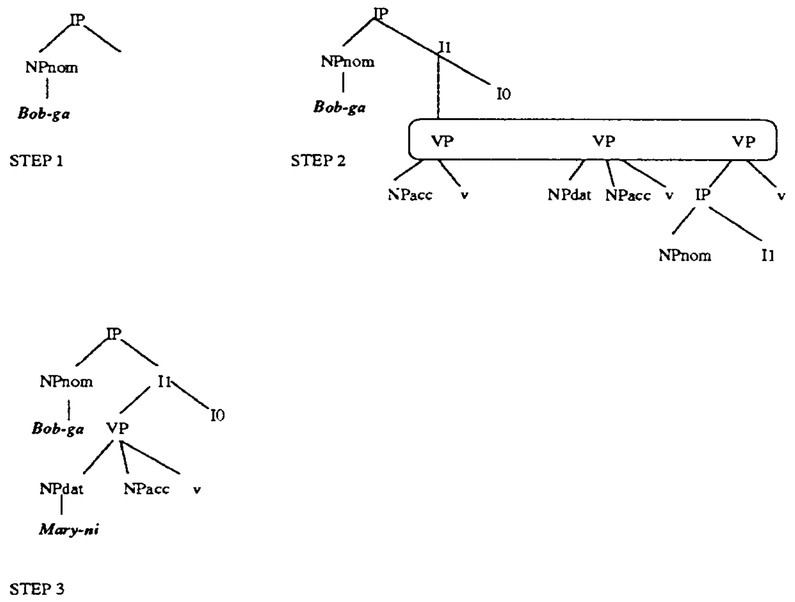


Figure 4.14
Parsing a head-final sentence with an AND/OR representation

on left recursive grammars, while at the same time having a strongly predictive behaviour, like top-down grammars. Notice that although a considerable amount of structure can be predicted, this algorithm cannot postulate structure beyond the specifier position of an embedded clause, thus immediately accounting for the surprise effect in sentence (65).

5 LOCALITY

Move-α, in current GB theory, is the only transformational rule. It is completely unrestricted, in the sense that it is interpreted as *move anything anywhere*. The underlying assumption is that this is the most general formulation of a movement rule, and hence the most explanatory. Thus the single movement rule of the theory, *per se*, does not incorporate any restrictions. The correct empirical description is achieved by imposing restrictions on the elements that move and on the source and target site of movement. Defining the domain of operation of move-α is thus crucial, but also complex.

Long distance relations between elements undergo strict locality conditions, as was noted as early as Ross (1967). Much work has been devoted, both in the linguistic and in the parsing literature, to the discussion of locality conditions on long distance dependencies.

In this chapter, after setting the theoretical framework (Cinque 1990), some alternative proposals in the parsing literature are reviewed that make crucial use of the notion of locality to restrict the amount of computation performed on-line and establish a strict relation between the parser and the grammar (Marcus 1980; Berwick and Weinberg 1984; Frank 1992). Drawing on their work, I shall discuss the treatment of cross-linguistic asymmetries. Finally, I shall discuss the implementation, arguing that preceding attempts to capture locality restriction in the architecture of the parser Berwick and Weinberg (1984) or Frank (1992)) are not successful, as they only capture one part of locality restrictions, namely that part that deals with the "minimal recursive subtree". I suggested that locality restrictions be encoded declaratively by pointers to the available left context.

5.1 THE LINGUISTIC FACTS

I shall restrict my attention to long distance dependencies created by *wh*-movement. *Wh*-movement presents two characteristics that are going to be of interest here: it undergoes locality restrictions, and it is subject to cross-linguistic variation.

Wh-movement is not unrestricted: extraction from certain constructions (called islands) gives rise to ungrammatical sentences. Some of the most revealing islands are exemplified below (Cinque 1990, 1ff).

(78) Subject Island
a. * Which books did [talking about t] become difficult?
b. * How would [to behave t] be inappropriate?

(79) Complex NP island
a. * To whom have you found someone who would speak t ?
b. * How have you found someone who would fix it t ?

(80) Adjunct Island
a. * To whom did you leave without speaking t ?
b. * How was he fired after behaving t ?

(81) Wh Island
a. ?? To whom didn't they know when to give their present t?
b. * How did they ask you who behaved t ?

(82) Negative Island
a. To whom didn't you speak t ?
b. * How didn't you behave t ?

(83) Factive Island
a. To whom do you regret that you could not speak t?
b. * How do you regret that you behaved t ?

(84) a. To whom is it time to speak t?
b. * How is it time to behave t?

Two facts can be noticed. First, in all cases extraction from islands generates some ungrammatical *wh*-movement. Second, sentences (81)-(84)a are grammatical, while (81)-(84)b are not. The islands that generate ungrammaticality in all cases are called ***strong islands***, while the others are called ***weak islands***. The fact that (81)-(84)a are grammatical shows that "long" movement is allowed in these cases, *i.e.* the long distance relation between the *wh*-word and the trace can be established. On the other hand, the ungrammaticality of (81)-(84)b shows that long movement is not available to the type of element that is being moved. However, movement is available to the same elements if it can occur in shorter steps, as the contrast in (85)-(87) shows.

(85) a. * How did they ask you who behaved t ?
b. How did they think you behaved t?

(86) a. * How didn't you behave t?
b. How did you behave t?

(87) a. * How do you regret that you behaved t?
b. How do you think that you behaved t?

These facts suggest that long *wh*-movement is sensitive to strong islands, while successive cyclic *wh*-movement is sensitive only to weak islands. Following Cinque (1990), then, three questions need to be answered with reference to *wh*-movement.

1. What classes of elements undergo long and successive cyclic movement?
2. From what principles of the theory does the existence of long and successive cyclic movement follow?
3. What is the nature of the locality conditions on long and successive cyclic *wh*-movement?

In Chomsky (1986a) the answer to the first and second questions is that arguments can undergo long movement, while adjuncts cannot.[1]

Refining on this proposal, Rizzi (1990) points out that some θ-marked complements, *i.e.* arguments, most notably measure phrases and NPs in idiom chunks, cannot undergo long movement, as shown in (88)-(89).

(88) * Quanti chili ti ha chiesto se pesavi?
"How many kilos did (s)he ask you if you weighed?"

(89) * L'attenzione che non ho ancora deciso a chi prestare, è poca.
"The attention that I have not yet decided to whom to pay, is little."

Rizzi (1990), therefore, distinguishes elements that receive a "referential" θ-role from those that receive a non-referential θ-role, such as a *measure* θ-role. The principle governing movement for non-referential elements, adjuncts, measure phrases and idiom chunks is the ECP, which imposes very local licensing conditions. The principles governing movement of referential elements are jointly the ECP and binding. The fact that referential elements can be "identified" by binding accounts for their ability to move longer distances than adjuncts. An even finer distinction is needed to account for the systematic difference in acceptability of the extraction of arguments related to a bare quantifier and extraction of NPs such as *tutti*-N, as shown in (90)-(91) (Cinque 1990).

(90) a. * Ogni dichiarazione mi chiedo perché abbia ritrattato.
" Every declaration I wonder why he repealed."
b. * Nessun libro mi domando perché abbia comprato.
"No book I wonder why he bought."
c. Tutti i musei, mi chiedo chi possa aver visitato.
"All the museums, I wonder who might have visited."

[1] This distinction is derived from a particular formulation of the ECP, such that intermediate traces of arguments may be deleted at S-structure: both arguments and adjuncts need to be antecedent governed, *i.e.* they can move only by successive short steps; however, in the case of arguments some of the intermediate traces left behind by these small steps can then be deleted, "simulating" long movement at the level of LF, while the intermediate traces of movement of adjuncts cannot be deleted.

WHICH-N	BARE-WH/NP
referential (D-linked)	referential (non-D-linked)
binding	antecedent government
long movement	successive cyclic movement

Table 5.1
Two types of wh-movements

(91) a. * Ogni museo, non vuole visitare.
"Every museum, he does not want to visit."
b. * Nessun libro non e' vero che abbia comprato.
"No book it is not true that he bought."
c. Tutti i musei, non ha visitato.
"All the museums, he did not visit."

If referentiality is interpreted as D-linking in the sense of Pesetsky (1987), referential means member of a preestablished set. Then, according to this definition, a bare *wh*-operator/variable configuration is not referential, and NP-traces are not intrinsically referential, if we think of them as elements of a discountinuos constituent. Cinque (1990) identifies then two kinds of operator/variable configuration, as shown in table 5.1, where the types of NPs that belong to the two classes, the grammatical principles that perform the *identification* of the empty category, and the type of movement are listed.

Within the *Barriers* framework, Cinque (1990) proposes to define two slightly different notions of barrier for long movement and successive cyclic movement.[2] Intuitively, the different definitions must capture the fact that strong islands are neither θ nor L-marked, while weak islands are θ-marked. The difference in the two cases can then be captured by the notion of *direct* marking, under sisterhood.

[2] Chomsky (1986a) proposes that the locality condition on successive cyclic movement is antecedent government, while the condition on long movement is Subjacency. Cinque (1990) points out that this proposal is unsatisfactory, since it deals with locality conditions in a way which is not uniform. Chomsky (1986a) attempts to unify the theory of government and the theory of bounding by using the same notion of barrier in both theories. Cinque (1990) notes that the unification is only partial, however, as a different number of barriers is relevant: one for government and two for bounding. Moreover, two more notions of barrier are needed in order to capture all the facts: minimality barrier for government and inherited barrier for Subjacency.

Definition

Barrier for Government (Cinque 1990, 42) (113)

Every maximal projection that fails to be directly selected by a category nondistinct from [+V] is a barrier for government.

Definition

Barrier for Binding (Cinque 1990, 42) (114)

Every maximal projection that fails to be (directly or indirectly) selected in the canonical direction by a category nondistinct from [+V] is a barrier for binding.

In sum, the trace of *wh*-movement must be formally licensed and identified. A nonpronominal empty category is formally licensed if the ECP, as formulated in 5.1, is satisfied. If the empty category is referential it is then identified if binding is satisfied, while a nonreferential nonpronominal empty category is identified iff antecedent government is satisfied.

Definition

ECP (Cinque 1990, 49)

A nonpronominal empty category must be properly head governed by a head nondistinct from [+V].

As it can be noticed, Cinque's definition of barrier reintroduces substantive notions, related to the feature [+V], which were absent from Chomsky (1986a)'s definition. The fact that the category label determines barrierhood, however, makes it impossible to use a parameterized list of bounding nodes. The cross-linguistic variation discussed in Rizzi (1982) is then basically stipulated. I review here the linguistic evidence and reinterpret Rizzi's proposal in the framework of the current theories of $\overline{\text{A}}$ movement.

5.1.1 Cross-linguistic Variation

Rizzi (1982, 49ff) notices that some of the evidence that motivated the Subjacency condition, namely the existence of *wh*-islands, does not translate directly into Italian.[3] Specifically, Italian could violate some *wh*-islands.

An example of long distance movement that is allowed in both languages is given in (92).

(92) a. $[_{CP}$ *Who*$_i$ do $[_{IP}$ you think $[_{CP}$ t_i that $[_{IP}$ Mary said $[_{CP}$ t_i that $[_{IP}$ Bill saw t_i ?]]]]]

b. $[_{CP}$ *Chi*$_i$ $[_{IP}$ credi $[_{CP}$ t_i che $[_{IP}$ Maria abbia detto $[_{CP}$ t_i che $[_{IP}$ Gianni ha visto t_i ?]]]]]

Both Italian and English, on the other hand, exclude *wh*-extraction from a *wh*-clause which is not the most embedded one, as shown in (93).[4]

(93) a. * Il mio primo libro, $[_{CP}$ *che*$_j$ $[_{IP}$ so $[_{CP}$ a *chi*$_i$ $[_{IP}$ credi $[_{CP}$ t_i che $[_{IP}$ abbia dedicato t_j t_i]]]], mi è sempre stato molto caro.

b.* My first book, which I know to whom you believe that I dedicated, has always been very dear to me.

The next two examples show the kind of configurations in which Italian can violate *wh*-islands: if the most embedded clause is a *wh*-clause, then extraction is possible and the extracted element can either move higher up the tree or move to the adjacent clause as shown in (94) and (95).

(94) a. Il mio primo libro, $[_{CP}$ *che*$_j$ $[_{IP}$ credo $[_{CP}$ t_j che $[_{IP}$ tu sappia $[_{CP}$ a *chi*$_i$ $[_{IP}$ ho dedicato t_j t_i]]]], mi è sempre stato molto caro.

b. * My first book, which I believe that you know to whom I dedicated, has always been very dear to me.

[3]The *wh*-island constraint, as stated in Ross (1967), says that no *wh*-element can be displaced out of a *wh*-constituent.

[4]These sentences are Rizzi (1982, 56) (18b,a) and 50 (6b), respectively.

(95) a. Tuo fratello, $[_{CP}$ a *cui*$_i$ $[_{IP}$ mi domando $[_{CP}$ che *storie*$_j$ $[_{IP}$ abbiano raccontato t_j t_i]]]], era molto preoccupato.

b. * Your brother, to whom I wonder what stories they told, was very worried.

In order to account for this cross-linguistic asymmetry, Rizzi reformulates Subjacency as a parameterised condition. In the formulation before Chomsky (1986a), Subjacency would hold in (96)b given (96)a.

(96) a. $\ldots \alpha \ldots [_{\beta} \ldots [_{\delta} \ldots \gamma]] \alpha \ldots$
b. No rule can relate α and γ in (96)a if two bounding nodes are intervening.

This restriction also applies to *wh*-movement. The result is that no single step in a cyclic application of movement could cross more than one bounding node. The bounding nodes are constants with the stipulated values of IP and NP for English.[5] Rizzi suggests that the value of the clausal bounding nodes can vary from language to language. He shows that by choosing the bounding node as CP for Italian the evidence above is easily explained. In no case, more than one CP bounding node can be crossed in Italian. Cinque (1990) notices however, that cross-linguistic variation is sensitive to referentiality. Those languages that allow *wh*-islands violations do so only with referential *wh*-phrases, or in relative clauses. Moreover, given this distinction, languages like English can also violate *wh*-islands given the appropriate context. Cinque gives the following examples.[6]

(97) (Cinque 1990, 53) (144a.b)
a. A car that I wouldn't know who to ask how to fix t.
b. These are the only vegetables which I don't know where to find out how to plant t.

In order to capture the cross-linguistic variation a stipulation is needed, which determines which node is an inherent barrier in a given language, along the lines

[5]The actual labels used in Rizzi (1982) to refer to sentential nodes are S and S'. I use here the terminology introduced by Chomsky (1986a), because it is just a label substitution. No matter of content is altered. CP = S' and IP = S.

[6]Cinque acknowledges that (97)a is from Browning, and (97)b is from Frampton (1990).

- The binder of a nonpronominal empty category must be the minimally local binder.
- The identification of a nonpronominal empty category is sensitive to D-linking properties of the empty category.
- Different locality domains can ensue different referential properties of the empty category.
- NP-movement generates a non-referential discontinuous constituent. Hence, nonreferential items and NP-movement must be treated in the same way.
- Cross-linguistic variation is sensitive to referentiality.

Table 5.2
Generalisation about wh-movement

of Chomsky (1986a, 37). Italian would then stipulate that the most embedded tensed CP is an inherent barrier for binding, while in English IP would count as an inherent barrier.[7]

Table 5.2 summarises the main generalisation about *wh*-movement

[7]This difference in the formulation of the parameter from the formulation preceding Chomsky (1986a) is in no way going to affect the validity of our discussion of Marcus (1980) and Berwick and Weinberg (1984) below, which were developed with a substantive theory of bounding nodes.

5.2 RELATED WORK

From the point of view of efficient parsing, unbounded dependencies pose a problem, because they require access to a potentially unbounded amount of left context. However, the theory of grammar, as formulated in GB, provides a tractable representation: every long distance movement is interpreted as the sum of a number of shorter steps. A deterministic parser is able to perform this computation, provided that the amount of material that every step can encompass is available as left context to the parser. In other words, the parser needs to know, given a *wh*-word as a cue of an upcoming trace, where to insert the trace.

5.2.1 Marcus 1980

Marcus (1980) adopts for his parser a *strictly deterministic* approach. He claims that a parser with the features in (98) is a deterministic parser.

(98)
1. No postulated node, no label and no grammatical feature are cancelled and no attachment is broken.
2. All syntactic structure created for a given input is part of the output tree.
3. No temporary encoding of syntactic structure is allowed.
4. Only limited lookahead is allowed.

Marcus (1980) uses two main data structures: a buffer, which can contain upcoming input tokens and assembled constituents whose attachment is not yet clear (thereby serving as lookahead device and temporary storage); and a look-into stack, which is a stack where elements other than the top one can be examined (thereby encoding the scanned portion of input to the left of the current token). The left context is limited to the *current cyclic node*, namely the current IP or NP.

The operations on these two data structures are limited by constraints on the order of insertion and retrieval of constituents, the *lowering lemma* and the *left-to-right constraint*, given in 5.2.1 and 5.2.1 respectively, from Marcus (1980, 141).

The Lowering Lemma The only reasonable method for lowering a trace bound to an NP in one clause into a lower clause is to do so implicitly by dropping the trace into the buffer.

The Left-to-Right Constraint The constituents in the buffer are (almost always) attached to higher level constituents in left to right order, *i.e.* the first constituent in the buffer is (almost always) attached before the second constituent.

The determinism hypothesis forces the parser to be a *wait-and-see* parser, in the sense that it cannot afford to make any mistakes, thus it can only postulate structures and attachments that are certain.

Marcus (1980) claims that it is precisely this hypothesis, coupled with the restrictions on licit operations, that make the parser obey a linguistic condition like Subjacency. For example, consider the parse of a long distance *wh*-movement. At the sight of a *wh*-word, the parser drops a trace into the buffer and a new sentential node is postulated. Before attaching the IP node, the parser builds a Comp node, which is formed by a complementizer word and trace. Then it attaches the Comp node as the first element of the newly postulated IP. As a consequence, when all the input is expended, the parser has built an $\bar{A}$-chain where every link in the chain, excluding the foot, sits in a Comp node. The restrictions on the available left context ensure that this is the only kind of cyclic movement allowed. Thus, *wh*-island behaviour is simulated. If one of the Comps that are encountered while parsing is already filled by a *wh*-word, then no trace can be dropped in the same Comp. Since the visible portion of the stack is the *current cyclic node*, at this point in the parse the Comp of the current cyclic node will contain no trace. Hence the information that a *wh*-word has been seen before is lost. Thus, when the A-position to drop the trace is finally found, no trace in Comp is available to bind it and yield the correct interpretation.

If we look at the Marcus parser from the point of view of principle-based parsing and precompilation which we have pursued so far, we see that the constraints on movement of the theory of grammar are not directly stated in the parser, but rather they are *folded* into complex structure-building actions. This is necessary because by precomputing the interaction between the generator principle move-α and the constraints, the parser acts deterministically. However, the efficiency of determinism is bought at the cost of cross-linguistic coverage, or even empirical coverage for English, in those dialects where *wh*-islands violations are acceptable (see for instance, Grimshaw (1986)).

The account for Italian is not so straightforward. In the specification for an algorithm that computes the Italian type of Subjacency, CP constituents belong to two classes: the class of those CPs that do not contain the trace in A-position and the class of those that do. This results from choosing CP as a bounding node. From the parser's point of view this means that a trace must be inserted in all Comps intervening between the antecedent and the trace, except the Comp immediately dominating the trace.

For a parser like Marcus (1980) to be able to compute Subjacency for Italian, it would need to recognise that the upcoming clause is the one that contains the trace in the base position. If the parser could recognise that it is at this stage in the parse, then it would know that it need not drop a trace in Comp. The amount of available left context would also vary, depending on the derivational cycle, to ensure proper binding of the trace. When the Italian-like parser arrives at the most embedded Comp it might not need to drop a trace, but then it needs to have access to the previous Comp to bind the trace in A position properly. Algorithm 1 would be able to capture this state of affairs.

Algorithm 1

```
1. If most embedded clause then
           left context ← from A-trace to closest $\bar{A}$-trace
   else
           left context ← current clause
2. If the left context contains a wh-word or a trace in Comp then
           if not most embedded clause then
                   insert trace.
3. No trace can be inserted in a wh-filled Comp.
```

Algorithm 1 parses Italian, but it would violate both the lowering lemma and the left-to-right constraint, since, according to the given definition of left context, a constituent could be built before retrieving a trace from the buffer. One could try to parameterise the Marcus algorithm to capture Italian, but this attempt would either violate the spirit of Marcus's approach or miss the explanation for English. Consider, for instance, a rather natural parameterization: the active node stack is parameterised. In Italian the value for the cyclic node could be CP, in English IP. This choice would reflect rather transparently the grammatical parameter and be faithful to the parsing design, but it would not work: as long as the lowering lemma and the left-to-right constraint are active, only one sentential node can be built when a trace is in the buffer. Basic *wh*-island violations in Italian would not be captured. Thus the mechanism would still be too strong.

On the other hand, if the lowering lemma and the left-to-right constraint were lifted, empirical problems would result for English: the mechanism would be too weak. Since the lowering lemma and the left-to-right constraint are crucial to enforce the Specified Subject Constraint (SSC), it is easy to predict that some ungrammatical sentences could now be parsed. Consider, for instance, the case of object raising below.

(99) * Mary seems $[_{IP}$ John to like t]

Object raising is excluded in Marcus (1980) parser because the conjunctive application of the two lemmas allows extraction only from subject position, but if the lemmas were lifted, then the trace could be created and bound to *Mary* before creating the IP node, and inserted later.

If the two constraints were parameterised, such that English enforces them and Italian does not, then Comp to Comp movement of *wh*-traces in Italian would be totally accidental. But Italian performs cyclic movement as is shown in (93) above. Thus Marcus (1980) is not easily extensible to capture cross-linguistic variation.

5.2.2 Berwick and Weinberg 1984

Berwick and Weinberg (1984) develop a two stage parser that consists of a tree building device and a coindexing procedure. The first stage of the parser, discussed here, draws on Marcus (1980).

Berwick and Weinberg (1984) claim that the Marcus parser is "an informal machine version of an LR(k) parser, specifically, a bounded context parser (p.153)", since it satisfies three defining properties of an LR(k) parser.

- It computes left to right a rightmost derivation in reverse.
- It is deterministic, in that it uses a finitely bounded lookahead.
- The parsing rules are stored in a finite control table.

By adopting this point of view, Berwick and Weinberg (1984) are able to offer a reason why apparently unrelated phenomena of linguistic theory obey the same constraints. These phenomena form a class for the parsing mechanism. Such constructions are *wh*-movement constructions, parasitic gaps and some kinds of gapping. (Berwick and Weinberg 1984; Berwick and Weinberg 1985; Fodor 1985).

The way their parser provides an analysis for cyclic *wh*-movement is slightly different from Marcus (1980)'s. They assume that a trace in an A-position is postulated depending on the θ-grid of the verb. If an antecedent trace is present in the left context then coindexation can occur. The antecedent trace is distinct from that of Marcus (1980), because it does not have to be attached at the moment the new clausal node is postulated. It can be inserted later, if no A-trace has been seen, provided it is inserted before the current domain has not been passed to the second stage device, where it is no longer visible. As a result, this parser provides the kind of design that can easily accommodate a parameterised version, because the action that builds the structure (*i.e.* inserts the trace) and the checking of the constraints are not bundled in a complex operation. We will present below the specification of an algorithm that works both for English and Italian. We note, however, that the formal properties of this parser are not exactly as claimed in Berwick and Weinberg (1984). First, the Marcus parser is not LR(k), but rather LRRL(k), as proved in Nozohoor-Farshi (1986). Intuitively, an LR(k) parser can only use terminal symbols as lookahead, while an LRRL(k), which stands for LR Fully Reduced Lookahead of k, can also use nonterminals. It is easy to see that the use of the buffer in the Marcus parser, which can contain fully-built unattached subtrees mimics an LRRL machine. Also, it is not a bounded context parser since a bounded context parser $BC(m,n)$, only takes the last m tokens into account, whereas the Marcus parser uses a packeting mechanism of pattern-action rules to encode the left context. For a more detailed explanation and proofs, see Aho and Ullman (1972) and Nozohoor-Farshi (1986). Second, as Van de Koot (1990) observes, the behaviour that mimics Subjacency is actually derived from the fact that a bound on feature annotation is imposed. In order to keep track of the current cyclic node, a deterministic parser can encode the information as a feature on each node. Thus, there is nothing intrinsic in the annotation that prevents to annotate more than one feature at a time. The parser obeys *wh*-islands as a consequence of the limit on feature annotation. Moreover, the only locality restriction that is derived is $\bar{A}$-minimality. However, we show below that the fact that the constraints are applied separately from structure building can be used to model cross-linguistic variation.

5.2.3 Frank 1992

Frank (1992) notes that locality restrictions are pervasive in GB theory. Every principle of the grammar is stated as holding only in a given local environment. Therefore, Frank argues, locality restrictions should not be stated explicitly, and redundantly, in the theory of grammar, but rather they should descend as a property of the grammatical formalism. Thus he proposes to consider grammatically local environments that can be defined over an elementary tree, as defined in Tree Adjoining Grammar (TAG). Frank suggests that the parser incorporates the theory of grammar directly (in fact, he allows no grammar precompilation), and that by restricting the operations of the parser to those operations that are defined over an elementary tree, the working space of the parser is guaranteed to be always parcelled into subtrees of bounded size, since by definition no recursion is encoded in the ETs. This parser would then proceed as follows. For every input token, it projects $\bar{X}$ structure and lexical information; following the lexical specifications to license phrase structure it attempts to incorporate the projection into the partially constructed tree. When the structure already built corresponds to an ET, either initial or auxiliary, then the ET is excised from the structure by an operation which is the reverse of the two operations defined in TAG to combine trees. Hence, an auxiliary tree is unadjoined and an initial tree is unsubstituted. The ET is then passed to a semantic component for interpretation.

Using TAG to define formally the amount of locality over which linguistic principles can span is appealing. However, some empirical issues remain to be settled. There are cases in which the definition of locality given by TAG and the locality restrictions defined in GB are not coextensive. For instance, according to the theory of locality developed at the beginning of this chapter, there are at least two locality domains for the identification of empty categories, depending on the content of the empty category: binding for D-linked expressions and antecedent government for non-D-linked elements. In the case of "long movement", namely extraction of arguments from weak islands, the notion of ET and the notion of local domain in GB seem at odds. For instance, take the example (81)a, repeated here for convenience.

(100) ? To whom didn't they know when to give their present?

Here the long binding required to bind the trace spans over a recursive structure. However, the sentence is at worst marginal. Thus TAG might impose a type of locality that is descriptively too restrictive, as it fails to provide definitions of locality for different referential items. The algorithm that uses TAG then suffers from the same shortcoming as the algorithms seen above. Namely the attempt to reduce locality to some sort of undoing of recursion is bound to be only partially successful, unless item-dependent and language-dependent differences are taken into account.[8]

In sum, precompilation of locality restrictions into the structure building routines is only partially successful. TAG might offer a well-formalized theory of locality domains provided it were enriched with complex symbols, or precompiled atomic symbols, that give rise to several different locality domains for each element type (A, $\overline{A}$, head, referential) and also provided that an efficient algorithm to consult the forest of ETs only at the relevant steps were specified.

5.3 PARAMETERISED SUBJACENCY

As a solution to the shortcomings of the deterministic algorithms seen so far, I propose a parameterised version of Berwick and Weinberg (1984), Berwick and Weinberg (1985). The algorithm is built on the following assumptions:

1. The parser can access only a limited amount of left context, called $\lambda(x)$.
2. The amount of left context can acquire different values, (*i.e.* it is a parameter).
3. Empty categories are formally licensed by principles of the grammar, such as the ECP.
4. After being licensed, a trace must be identified by an antecedent.
5. Given Subjacency, the antecedent for a trace must be in $\lambda(x)$.

[8]Maybe TAG can be used if complex node labels on trees are used to determine recursive structure. We need then to develop a theory of complex nonterminal symbols and to investigate the complexity of algorithms that use them.

Algorithm 2

```
1.  fix bounding node parameter

2.  scan sentence:
        repeat until end of input
        update $\lambda(x)$
        if Comp = not c-ommanded by real gap chain then
            drop empty operator
        if Comp = - wh then
            drop trace
            bind trace
        scan CP

3.  scan CP:
        while trace in IP do
        if trace in A-position is licensed then
            drop trace
            look for antecedent in $\lambda(x)$
            if antecedent available then
                bind trace
            else fail
        else fail
```

Figure 5.1
Parameterized Algorithm to Compute Subjacency

6. Doubly filled Comps are not allowed.

If we fix $\lambda(x)$ = adjacent x, and $x \leftarrow$ IP for English, and $x \leftarrow$ CP for Italian, then the correct array of evidence can be derived. We assume that the execution of this algorithm is triggered by the detection of a *wh-environment*, questions or relative clauses, for instance. At the moment it is also assumed that such environments are identified by *wh*-words or relative pronouns.[9] The algorithm is shown in Figure 5.1.

[9]Thus we assume that an Active Filler Strategy is at work (modulo De Vincenzi (1991), see chapter 4).

5.3.1 Explanation and Comments

Step 1 sets the bounding node parameter for the current language, while Step 2 is the main loop that guarantees that the parsing of multiclausal sentences is performed as a collection of clausal parses.

Step 3 assumes that traces in A-position are licensed by proper head government structure. When a trace is postulated it must be identified by an antecedent contained in $\lambda(x)$. Since the trace is dropped first and then the left context is scanned for the binder, at this point the algorithm is going to take advantage of the possible different value of $\lambda(x)$ to derive the asymmetry observed in the data. If $\lambda(x)$ is the adjacent CP, the Italian value, this means that the trace in A-position can look at two Comps for its antecedent. As a result, even if the nearer Comp is filled, the trace can still be bound and the sentence is grammatical. If $\lambda(x)$ is the adjacent IP, as in English, then only one Comp node is available to identify the trace in argument position. Either the Comp is an eligible identifier or the sentence is excluded.

The condition that drops an empty operator in Comp is needed to parse parasitic gaps, and it is based on the discussion in Berwick and Weinberg (1985). Parasitic gaps are problematic for a deterministic parser because they are in complementary distribution with overt pronouns, as can be seen in (101).

(101) a. Who did you meet without greeting ?
b. Who did you meet without greeting him ?

Parasitic gaps are required to be bound by an empty operator and moreover, they obey Subjacency (Chomsky 1982; Chomsky 1986a). The parser then must be able to predict a parasitic gap to be able to build a correct chain of $\overline{A}$ links to bind the gap. For example, the representation for the sentence in (102) is as follows.

(102) *Who*$_i$ did you meet t_i O_i without greeting e_i ?

The distribution of parasitic gaps obeys conditions that are not completely clear. Parasitic gaps can be attached either as adjuncts or as subjects, as exemplified in in (103) and (104) below; they show some *island* effects, hence obey subjacency; and they are licensed at S-structure.

(103) Adjunct Parasitic Gaps
a. What did you file t [before reading e ?]
b.* Who [$_{IP}$t met you [before you recognised e ?]

(104) Subject Parasitic Gaps
a. A man who [whenever I meet e] [t looks old]
b.* A man who [t looks old [whenever I meet e]

As mentioned above, according to Chomsky (1982),Chomsky (1986a) an empty operator in the Comp licenses a parasitic gap. Thus two chains are present and the licensing of parasitic gaps can be reduced to a condition that regulates chain composition. (See above, chapter 4.) A plausible candidate is anti-c-command. Anti-c-command requires that the parasitic gap be not c-commanded by either the real gap or the real operator. Consider then anti-c-command as the condition on chain composition. From a parsing point of view it is really not surprising that chain composition should be subject to this kind of restriction. If we assume that Subjacency incorporates the notion of c-command, then whenever an $\bar{A}$-chain is being computed and a new not c-commanded structure is postulated, the parser knows that a new $\bar{A}$-chain should be started.[10] The restriction on how much $\bar{A}$-chain stacking is possible could be regulated by the same mechanism that does not allow triple center embedding in natural languages (*cf.* Miller and Isard (1964)).

The parser then can drop the empty operator in Comp when the chain composition condition is satisfied, as suggested by Berwick and Weinberg (1985). Empty operators are not visible to the semantic component unless they bind an argument position by the time they are passed to the semantic interpreter. This is because they need to bear a θ-role to be visible and to receive a semantic index. By adopting this algorithm, a deterministic parser is able to be equally well equipped to parse a parasitic gap or a pronoun.

If the pronoun is in the position of the parasitic gap then the empty operator never enters a chain and does not receive a θ-role, so it is invisible to the semantic interpreter. Moreover, since the empty operator is in the head of the adjunct and the parasitic gap must be subjacent to it, an asymmetry between Italian and English is predicted. The parameterised algorithm must be able to deal with it and it does. The two following sentences provide the empirical evidence.

[10]For other reasons to include c-command in the definition of Subjacency, *cf.* Weinberg (1988).

STEP	STATEMENT	SET OF ACTIONS	CURRENT STATE OF THE INPUT
1.	fix parameter	$\lambda(x) \leftarrow$ CP	
2.	scan sentence	Comp = $+wh$ scan CP	... $[_{CP}$ *che*$_j$ $[_{IP}$ credo
3.	scan CP	no trace in IP	
2'.	scan sentence	Comp = -wh drop trace in Comp	$[_{CP}$ *che*$_j$ $[_{IP}$ credo $[_{CP}$ t_j che
3'. 2*.	scan CP scan sentence	no trace in IP ⋮ Comp = $+wh$ scan CP	
3*	scan CP scan CP	if a trace is licensed then drop trace $wh_i \in \lambda(x)$ then available antecedent bind trace if a trace is licensed then drop trace $wh_j \in \lambda(x)$ then available antecedent bind trace	$[_{CP}$ *che*$_j$ $[_{IP}$ credo $[_{CP}$ t_j che $[_{IP}$ tu sappia $[_{CP}$ a *chi*$_i$ $[_{IP}$ ho dedicato t_i $[_{CP}$ *che*$_j$ $[_{IP}$ credo $[_{CP}$ t_j che $[_{IP}$ tu sappia $[_{CP}$ a *chi*$_i$ $[_{IP}$ ho dedicato t_i t_j

Figure 5.2
Snapshot of Good Italian Sentence: *Il mio primo libro, che credo che tu sappia a chi ho dedicato, mi è sempre stato molto caro*

(105) ?? *Who*$_i$ did you meet t_i $[_{CP}$ O_i before $[_{IP}$ asking John $[_{CP}$ when $[_{IP}$ Sue would get married to e_i ?]]]]

(106) Il *ragazzo*$_i$ che Maria ha sposato t_i $[_{CP}$ O_i prima che $[_{IP}$ io avessi tempo di chiedermi $[_{CP}$ se $[_{IP}$ potesse conoscere e_i bene.]]]]

We now go through snapshots of the the algorithm to see it work in detail. A sentence which is good in Italian ((81)) and bad in English ((94)) and one which is bad in both languages ((93)) is illustrated.

STEP	STATEMENT	SET OF ACTIONS	CURRENT STATE OF THE INPUT
1.	fix parameter	$\lambda(x) \leftarrow$ IP	
2.	scan sentence	Comp = $+wh$ scan CP	... $[_{CP}$ *which*$_j$ $[_{IP}$ I believe
3.	scan CP	no trace in IP	
2'.	scan sentence	Comp = -wh drop trace in Comp scan CP	$[_{CP}$ *which*$_j$ $[_{IP}$ I believe $[_{CP}$ t_j that
3'. 2*.	scan CP scan sentence	no trace in IP Comp = $+wh$ scan CP	
3*	scan CP	if trace licensed drop trace $wh_i \notin \lambda(x)$ fail	

Figure 5.3
Ungrammatical English Sentence: *My first book, which I believe you know to whom I dedicated, has always been very dear to me*

STEP	STATEMENT	SET OF ACTIONS	CURRENT STATE OF THE INPUT
1.	fix parameter	$\lambda(x) \leftarrow$ CP	
2.	scan sentence	Comp = $+wh$ scan CP	....$[_{CP}$ *che*$_j$ ...
3.	scan CP	no trace in IP	... $[_{CP}$ *che*$_j$ $[_{IP}$ so $[_{CP}$ a *chi*$_i$..
2'.	scan sentence	Comp = $+wh$ scan CP	
3'.	scan CP	no trace in IP	..... $[_{CP}$ *che*$_j$ $[_{IP}$ so $[_{CP}$ a *chi*$_i$ $[_{IP}$ credi....
2".	scan sentence	Comp = -wh update $\lambda(x)$ drop trace fail	

Figure 5.4
Ungrammatical Italian Sentence: *Il mio primo libro che so a chi credi che abbia dedicato, mi è sempre stato molto caro*

The boundedness of the left context explains why Italian can cross only the most embedded *wh*-word and not *any* Comps in any sentence. The algorithm is flexible enough, however, to allow double *wh*-extractions from the most embedded sentence.

Algorithm 3 allows a wider left context than either Marcus (1980) or Berwick and Weinberg (1984) do, but it maintains the explanation of why that particular amount of left context is chosen. Natural languages do not have counters. This property is usually given as the explanation for the fact that the adjacent clause constitutes the relevant left context. The only locality predicate that can be stated without using counters is *adjacency*. Algorithm 3 then expands the left context in a way that captures the data, but still uses predicates that belong to the vocabulary of natural languages.

5.4 IMPLEMENTATION

The lesson that we draw from the previous discussion is that attempts to derive locality constraints from the architecture of the parser or from the metagrammar need to be refined. It was shown that most proposals can derive rather elegantly some kind of *closest binder* requirement, since that is directly related to the notion of recursion on some linguistic entity. However, they all fail to capture the totality of the locality requirements. This is because recent developments of linguistic theory have formulated different locality restrictions for different input tokens. Thus in the same sentence, several, separate locality domains can be active. A descriptively adequate parser must be able to express this fact. We believe that both Berwick and Weinberg (1984) and Frank (1992) could be adapted. We present here our implementation of locality restrictions, which is a refinement of Berwick and Weinberg (1984), in that it operates on the stack of an LR parser.

Since there is no single locality constraint for movement, we propose to think abstractly of the parsing mechanism as operating on a *family* of stacks, each of which is relevant for different types of input tokens. Given the linguistic theory that we have presented above, two types of barriers are defined and three different minimality requirements. We summarize them in Figure 5.5. These locality conditions could be considered a theory of left context delimiters. What emerges is that the theory takes the *content* of the delimiter into account.

PRINCIPLE	INPUT TOKEN (TYPE)	ISLAND
Relativized Minimality	$\bar{A}$	*wh-island*
Relativized Minimality	A	superraising, superpassive
Relativized Minimality	head	*that*-t
Barrier for Government	non-D-linked	long distance extraction
Barrier for Binding	D-linked	long distance extraction

Figure 5.5
Locality Principles

This is apparent if we reformulate locality restrictions from the point of view of the parser. The locality requirement expressed by Relativized Minimality can be easily abstracted from the type of input, to a general formula (as it is actually expressed in Rizzi (1990)). We show in (107) one of the possible ways of specifying it.

(107) The available left context for X is that portion of the tree from the current input token of type X up to the closest element of type X (*i.e.* the maximal nonrecursive subtree on X).

The formulation of *relative minimal binder* has two properties that differ from that of *barrier*. First, the type of the relative minimal binder is determined by the input, *i.e.* the definition of left context is a parameterized function. Second, the parameter is related to heterogeneous notions, according to our IC Classes, as it distinguishes heads, which are considered a purely configurational notion, from A and $\bar{A}$ positions. A/$\bar{A}$ positions are usually listed exhaustively, and they depend on the configuration (*i.e.* all adjoined positions are $\bar{A}$), but also on the categorial status of the head of the maximal projections. For example, the specifier of I is an A position, while the specifier of C is an $\bar{A}$ position. On the other hand, there is no obvious correlation between the definition of barrier and the elements for which a given barrier functions as left context delimiters. Given the definitions of barrier for government and barrier for binding above, we could define the left context analogues as below.

(108) The available left context for X, X non-D-linked, is the portion of the tree from the current token X, up to the first barrier for government.

The properties of a barrier for government are that it is a maximal projection and that it is not directly selected from a category non-distinct from [+V]. Conversely, a non-barrier for government is either a non maximal projection, or a maximal projection which is a sister to V or I or C, and which is selected by it. A barrier for binding is defined analogously.

Since the locality restrictions have no *functional* relation to the architecture of the parser, we propose that they be stated explicitly in the parser. Since several types of locality domains need to be checked for each element, we might want to precompute the licit interactions and eliminate those interactions that are never going to occur for independent reasons.

Two kinds of restrictions are not going to arise: the conjoined satisfaction of A-minimality and binding locality and head-minimality and binding locality. In other words, take the locality conditions to be constraints on movement, that must be satisfied by an empty category to be identified. They could be expressed by the following six conjunctive statements.

(109) a. locality ← head-minimality, no-barrier-for-binding.
b. locality ← A-minimality, no-barrier-for-binding.
c. locality ← $\bar{A}$-minimality, no-barrier-for-binding.
d. locality ← head-minimality, no-barrier-for-government.
e. locality ← A-minimality, no-barrier-for-government.
f. locality ← $\bar{A}$-minimality, no-barrier-for-government.

The first two however will never be relevant, since heads and A-chains are not referential items in the sense relevant here, namely D-linked. Thus they cannot be licensed by binding. We are then left with the following locality constraints.

(110) a. locality ← $\bar{A}$-minimality, no-barrier-for-binding.
b. locality ← head-minimality, no-barrier-for-government.
c. locality ← A-minimality, no-barrier-for-government.
d. locality ← $\bar{A}$-minimality, no-barrier-for-government.

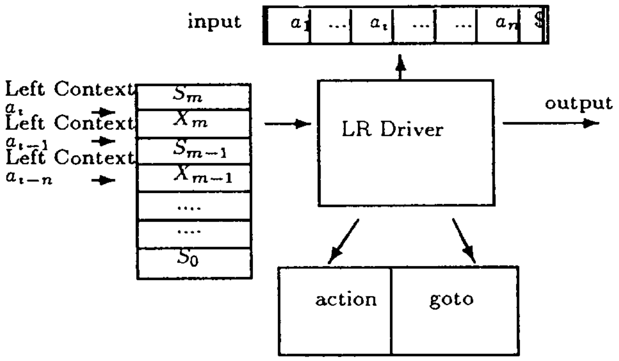


Figure 5.6
Family of Stacks for Left Contexts

Each of these conditions applies to different elements, as (110)a applies to referential(D-linked) $\bar{A}$ phrases, (110)b applies to heads, (110)c applies to phrases in A position and (110)d applies to non-referential (non-D-linked) $\bar{A}$ positions. Abstractly, we wa,' to keep track of a specific locality domain for each type of element.

The implementation is stiaightforward. A pointer to the relevant portion of the stack is maintained for each element that triggers the locality constraint. Whenever an empty category is posited by the LR algorithm, it is then formally licensed by the ECP, independently of its identification. The locality domain in which the empty category must be identified is then determined. A sentinel which is specific to that particular empty category limits the available left context in the stack. Pictorially, this is shown in Figure 5.6.

Thus, in principle, there is no limit to the number of categories of each type that are extracted. A limit *a priori* of the number of possible extractions appears to be empirically incorrect, as was argued in chapter 4 (*contra* Berwick and Weinberg (1984) and Correa (1988)) Locality restrictions are only partially a function of the type of element that is extracted (because of Relativized Minimality). They also depend on the feature ±V of the maximal projection intervening between the antecedent and the trace.

In the current implementation, an empty category simply triggers the *barrier* module. Now, notice that in an LR parser, when an empty category is postulated, the maximal projections between the trace and the antecedent have not been built yet. Therefore, the parser cannot immediately compute whether there are intervening barriers between the trace and the antecedent. On the other hand, the maximal non-recursive left context can already be computed. Hence, locality conditions are checked in two steps: first, the minimal local binder is found and the relevant left context is identified; then, the barrier constraint is *posted*. This means that new clauses are added dynamically to the program, which checks that the maximal projections intervening between trace and antecedent are not barriers. When all the intervening maximal projections have been built, and none of them is a barrier, then the constraint related to the licensing of a particular empty category is lifted and the category is identified.

With respect to considerations of a more general nature, this approach then does not endorse the functional view of the relation between the parser and the grammar (Berwick and Weinberg 1984); moreover, it does not impose any limit on the number of extractions; and finally, it determines dynamically the kind of computation to perform. In fact, the filtering principles related to traces are not applied unless traces are postulated in the phrase marker. More generally, filtering principles apply only if the relative generating principles are triggered. (See first observation on grammar principles in chapter 1.)

This design implies that the theory of grammar is taken to determine the parser at compile time, but that the parser at run time can vary depending on the actual input. Of course, the range of variation is determined by the compile time setup. This view is not at all dissimilar from the view implied in the principle-ordering approach of Fong (1991).

From the psycholinguistic point of view this approach obviously makes the claim that what is used on-line (and measured by experiments) is a dynamically changing object. Our proposal assumes that the criterion that determines the way in which the dynamic program is set up is a principle of minimum effort. Given a highly modular system, only the minimum amount of computation needed to accept or reject a sentence is performed. So, this approach is not entirely at odds with the principle-ordering approach, but it differs from it because the principle-ordering parser of Fong (1991) applies all the constraints to a grammatical sentence. We propose that not all principles are applied to all sentences. For instance, the locality principles that regulate the distribution of empty categories are only applied if there are empty categories in the sentence.

A

THE COMPUTATIONAL AND THE LINGUISTIC FRAMEWORK: A GLOSSARY

In this section the terminology is explained that is used in the main body of the work. The definitions are taken from Haegeman (1991) (abbreviated as H91), the page where the original definition appears is given.

A.1 LEVELS OF REPRESENTATION

A.1.1 The Levels

GB theory consists of several subtheories which interact with each other. These subtheories are hypothesized to operate at different levels of representation: D-structure, where elements occupy their grammatical function position (subject, object); S-structure, where elements are in different positions with respect to the level of D-structure; the level of phonological form, PF; and the level of logical form (LF), at which operator-variable and quantification constructions are interpreted. This organization of the theory is usually represented by the "Y model", shown in Figure A.1.

Move-α is a movement rule, which accounts for the mapping between levels. Consider the following example.

(A.1) Who does Mary like?

$$\begin{tikzcd}
 & \text{D-structure} \arrow[d, dash, "\text{move-}\alpha"] & \\
 & \text{S-structure} \arrow[dl, dash] \arrow[dr, dash, "\text{move-}\alpha"] & \\
\text{Phonological Form} & & \text{Logical Form}
\end{tikzcd}$$

Figure A.1
The Y Model

The sentence in A.1 is a question about the identity of some human whom Mary likes. It could be answered by *Mary likes John* or shortly *John*. To capture the fact that *who* refers to the object of the action of liking, two levels of representation are postulated, and "connected" by a movement rule. Thus the D-structure representation and S-structure representation of A.1 are A.2a,b respectively.

(A.2) a. Mary loves who.
b. Who does Mary love t_i?

In A.2b *who* has moved, leaving behind an "gap", called *trace*, which receives the thematic role and syntactic feature of an object. Moreover, the fact that *who* is an operator, in the sense that it binds the range of interpretation of the trace, is represented at a different level, the level of logical form.

D-structure

D-structure is a representation of lexical properties. D-structure representations are subject to the θ-criterion, which says that all arguments of a predicate must be present at D-structure.

S-structure

S-structure is related to D-structure by the Extended Projection Principle and by the rule move-α.

LEVEL	MODULE
Phonetic Form	Stylistic Rules
Logical Form	Operator-variable ECP Free Indexation Binding
S-structure	Theta Criterion Control Subjacency Case Filter
D-structure	Inherent Case $\overline{X}$

Figure A.2
Relation between Modules and Levels

Logical Form

This is the level at which sentences are interpreted. Quantifiers and *wh*-questions receive their scope at this level. LF is related to S-structure by an "invisible" version of move-α.

Phonetic Form

At this level lexical insertion and stylistic rules takes place.

A.1.2 Mappings between Levels

The Projection Principle

Lexical elements (words) play an important role in determining the structure of a sentence. Each constituent, *e.g.* NP, VP, AP, is projected from a head, a verb, a noun, an adjective respectively. Moreover, the internal structure of a sentence is in great part dependent on its thematic relations. In sum, the structure of a sentence is projected from the lexicon. This is true at all levels of representation. Thus, the Projection Principle determines the relation between the levels of representation.

The Projection Principle (H91:47)
Lexical information is syntactically represented.

The Extended Projection Principle

Sentence structure is mostly lexically determined. However, there is a general property of all sentences which is not always lexically represented, namely the fact that all sentences have a subject, whether it is lexically realized or not. Thus, this general property must be explicitly stated in the grammar.

The Extended Projection Principle (H91:59)
$S \rightarrow NP\ AUX\ VP$

A.2 THE MODULES

A.2.1 The Lexicon

The lexicon is a database of information related to individual words: for example, the category of the word, its argument structure, the way it is pronounced and its morphological properties. For example, the word *love* is of category Verb, it is diadic, *i.e.* it takes two arguments, an agent and a patient, it takes the auxiliary *have* in compound tenses, etc. Not all the information in the lexicon is completely idiosyncratic, in fact vast regularities have been studied. Indeed, if this were not the case, the lexicon would be very difficult to learn.

Category Features (H91:33)

Each word in the lexicon belongs to a category class. The category of a word can be determined by its distribution. The following classes are distinguished: Verb, Noun, Adjective, Preposition, Adverb, Inflection, Complementizer, Tense, Negation. Examples of each are: *love*, *dog*, *beautiful*, *from*, *quickly*, *-ed*, *whether*, *not*.

Subcategorization Frame (H91:34)

Besides belonging to the *category* Verb, verbs can be subdivided into *subcategories* depending on the complements they take. For example, *meet, imitate, love* all take an object, thus they are classified in traditional grammars as *transitive*. Verbs such as *sleep, die* do not take an object: they are intransitive. Verbs such as *believe* can take both a nominal object and a sentential object. These properties are encoded in the lexicon in a schematic way by means of *subcategorization frames*. Some examples are given below.

(A.3) a. *meet*: [_ NP]
b. *sleep*: [_]
c. *believe*: [_ NP/S]

C-selection

A piece of information that is encoded in a subcategorization frame is what category the complement of a verb must have. Thus, A.3a says that the complement of the verb *meet* must be an NP, and therefore * *I met to Sally*, where the complement is a PP, is an incorrect sentence. Thus, the verb selects the category of its complement. In this instance, the complement must be a noun.

Functional Selection

Functional selection is simply a more specialized term for c-selection, which is used if the selecting head is a functional category. I(nflection), C(omplementizer), and D(eterminer) are functional heads. Functional selection is a function, since I always selects a verb, C always selects I, and D always selects a Noun.

A.2.2 Configurations

$\overline{\text{X}}$ Rules (H91:95)

The rewrite rules in the phrase structure component can be described by a general schema, called the $\overline{\text{X}}$ schema.

(A.4) $X'' \rightarrow (Spec)\ X'^{*}$
$X' \rightarrow X\ \ Compl^{*}$

The notation is rather standard: * is the Kleene operator, for reflexive transitive closure (*i.e.* X* means that there could be 0 or more repetitions of X). Parentheses mean optionality.

This schema encodes the following information:

1. rules are endocentric: they are projected from a lexical head. For example, a verb phrase must contain a verb, and a noun phrase must contain a noun.
2. Rules are projections from a head up to a maximum number of levels. The maximal projection is 2. The level which is projected from the lexicon is called head, or zero-level projection.
3. Rules are unordered.

This notation is a shorthand for the rewrite rules that could be written for all categories. Thus X is a variable that ranges over all major categories. A well-formed phrase structure rule is $NP \rightarrow N'\ \ PP$, an ill-formed phrase structure rule is $NP \rightarrow V'\ \ PP$.

Sisterhood

According to the $\overline{\text{X}}$ schema, a complement is sister to the zero-level projection, and a specifier is sister to the level-one projection of a head. A node is ***sister*** to another node if they are ***immediately dominated*** by the same node in the tree, their *mother*. Node A immediately dominates node B in a tree, iff, on the path from B to the root, A is the closest node to B.

C-command (H91:125)

C-command is a configuration which is relevant in linguistic theory, although it is not primitive from the point of view of structural representation. It was shown by Reinhart (1976) that c-command, for c(onstituent)-command, is the relevant way to define coreference domains.

α c-commands β iff α does not dominate β and every γ that dominates α also dominates β.

In this case γ is the first branching node dominating α.

M-command (H91:125)

α c-commands β iff α does not dominate β and every γ that dominates α also dominates β.

In this case, γ is the first maximal projection dominating α.

λ Rules

These rules, that are usually never explicitly mentioned in the linguistic literature, are a necessary augmentation of the $\overline{\text{X}}$ schema in order to build the phrase marker for sentences with empty categories. They simply say that certain non-terminals can be substituted by the empty string. According to Chomsky (1986a, 4), only maximal projections and zero-level categories can be moved, *i.e.* they can be substituted by the null string at some level of representation.

A.2.3 Case Theory

Abstract and Overt Case

Consider the sentences *She looked at him*, and *He looked at her*. In the first sentence, the female subject is expressed by *she* while in the second sentence the female actor is expressed by *her*. Different morphological forms are used because in one case the word is the subject, while in the second it is the indirect object of the verb. This process of grammaticalizing the participants in an action is called *Case*. In English, case is overt only for pronouns, as one can see from the following pair. *The woman looked at the man*, *the man looked at the woman*. Other languages however, use this distinction much more extensively: German, Latin, Finnish, among many others.

For generality, however, it is assumed that every NP has a case. If the case is not overt, it is abstract (denoted by capital letter). The property of having or not having Case is syntactic as it affects the distribution of NPs in a sentence (and also of empty categories.)

Case Filter (H91:156)

Every overt NP must be assigned abstract Case.

Structural Case Assignment

A transitive verb or a preposition assigns ACCUSATIVE case, while nouns and adjectives do not assign Case. This accounts for the following sentences.

(A.5) a. The big ape imitated the man.
b. * He slept them.
c. He stared at the children.
d. The destruction of the city.
e. *The destruction the city.
f. His mother is proud of him.
g. *His mother is proud him.

The tensed morpheme of a verb assigns NOMINATIVE case, while an infinitive verb assigns no Case to its subject.

(A.6) a. I think that Mary is a fool.
b. * I think Mary to be a fool.

Inherent Case Assignment (Chomsky 1986a, 194)

If A is an inherent case assigner, then A assigns case to an NP iff A θ-marks the NP.

This means that the difference between structural and inherent Case assigment rests on their sensitivity to θ-role assignment.

DATIVE and GENITIVE in German are assumed to be instances of inherent Case. This difference can be shown by looking at passivisation, which affect only structural case assigment, as shown by the following paradigms.

(A.7) *Sie sieht ihn.*
she sees him-ACC

(A.8) *Er wird gesehen*
*Ihn wird gesehen
He-NOM is seen

(A.9) *Sie hilft him*
she helps him-DAT

(A.10) *Ihm wird geholfen*
*Er wird geholfen
He-DAT is helped

A.2.4 θ Theory

θ-roles

Using the idea of a function in formal logic, linguistic theory states that every predicate has a certain number of arguments. The arguments are the participants in an event. For example, in *John gave Mary a book*, there are three participants in the *giving* event: an agent (John), a receiver or goal (Mary) and the thing given or theme (the book). Notions such as agent, goal and theme are called thematic roles, or for short, θ-roles. The theory of thematic roles consists of two parts: one part that deals with the content of such roles, and one part that deals with their distribution and their relation to predicates. The former part is still very sketchy. The second part constitutes the bulk of θ-theory. Its main principle is the θ-criterion.

θ-Criterion (H91:46)

Each argument is assigned exactly one θ-role.
Each θ-role is assigned to exactly one argument.

In other words, the θ Criterion is a bijection principle between arguments and θ-roles. It guarantees that the thematic structure is mapped onto a well-formed structure. It rules out sentences where there are too many nouns, such as *John loves Mary Lucy*, or where there are too few nouns, such as *John puts*.

A.2.5 Movement Theory

Move α

John loves Mary and *Who does John love* are related, as the former could be taken to be the answer to the latter. This means that both sentences have the same function-argument structure. This fact is expressed by current linguistic theory by assuming that there is more than one level of representations for each sentence. In this example two are relevant: one at which the sentences are similar, and one at which they correspond to the superficial string. The two sentences receive the following "deeper" representation, where the similarity is apparent.

(A.11) a. John loves Mary.
b. John loves who.

The superficial level will be as shown in A.12.

(A.12) a. John loves Mary.
b. *Who$_i$* does John love t_i?

The fact that *Who* is the object of the verb is expressed by representing it both as the first word in the sentence, as is necessary to fit the string, but also as having the same index i as the empty slot, t_i after the verb. We say that *Who* has moved. There is a single transformational rule in current GB theory: move-α.

Move-α *Move anything anywhere.*

Coindexation

Two elements are coindexed if they are assigned an index and this index is identical. See for example, *Who does John love?* in A.12

Chains

Informally, a chain is a syntactic object that defines an *equivalence class of positions* for the purpose of feature assignments and interpretation. For example, take the passive sentence in A.13.

(A.13) *Mary$_i$* was loved t_i

The sentence in A.13 contains the chain *(Mary$_i$, t_i)*. Here t_i receives a θ-role from the verb, but no case, which is absorbed by the passive morphology, while *Mary$_i$*, receives nominative case because it is in a structural position that is inherently case marked, Spec of IP. This position, though, receives no θ-role, because of passive morphology again. The set of positions, however, satisfies the conditions on lexical argument NPs, namely one half of the θ-criterion and the Case Filter.

Chain A chain C $(E_1, ..., E_n)$ for n $\geq$ 1, is a sequence of elements in a phrase marker, where E_1 is the head of the chain and E_n is the foot of the chain.

Link A link of a chain L, (E_i, E_{i+1}) is an ordered pair of consecutive elements of the chain.

A Chains, $\overline{\text{A}}$ Chains

Chains can be classified according to the status of the landing site: A chains are those that are headed by an element in A (argument) position, shown in A.14; $\overline{\text{A}}$ chains are headed by an element in $\overline{\text{A}}$ (non-argument) position, such as A.15

(A.14) *Mary*$_i$ seemed e'_i to have been loved e_i

(A.15) *Who*$_i$ did John think e'_i that Mary loved e_i?

Multiple Chains

More than one chain can occur in a sentence. Multiple chains occurring in the same sentence can either be disjoint, intersected or composed. Disjoint chains are nested, as in A.16.

(A.16) *Who*$_i$ did *Mary*$_j$ seem t_j to like t_i

If chains intersect they share the same index and they have exactly one element in common.

(A.17) *Who*$_i$ did you think e'_i t_i seemed t_i to like Mary?

If chains compose they don't have intersecting elements, but they create a new link: if E_n is the foot of one chain and E'_1 is the head of the following chain then $< E_n, E'_1 >$ is a link in the composite chain. This is exemplified in A.18.

(A.18) *Who$_i$* did you meet t_i O_i without greeting t_i ?

Superindexed Chains

Some languages, for example Italian and Spanish, can postpone the subject to the end of the sentence. These constructions are usually analysed as involving a superindexed chain, namely a chain that differs from subindexed chains because the elements that form it are not generated by movement. We discuss here some of the properties of such chains.

Inverted Subjects In some languages, like Italian, the subject of the sentence can occur before the verb, or be inverted or be null.

(A.19)	*Gianni*	*ha*	*telefonato.*
	Ha	telefonato	Gianni.
	Gianni has called.		

(A.20)	*pro*	*Ho telefonato*	*a*	*casa.*
	I	called	home	

Subjects of intransitive verbs belong to two different classes: subjects of unaccusatives and subjects of unergatives. Several important pieces of work have dealt with unaccusative verbs in Italian (Burzio 1986; Belletti and Rizzi 1981; Belletti 1988). We discuss here very briefly the different structural position of the two classes of subjects.

Unaccusatives The subjects of unaccusative verbs behave like structural objects in many respects. In Italian they can undergo *ne* extraction and they trigger past participle agreement. Both these diagnostics are assumed to indicate movement from structural object position.

(A.21)	*Ne*	*ho visti*	*molti.*
	of-them	I saw	many
(A.22)	*Ne*	*sono arrivati*	*molti.*
	of-them	arrived	many
(A.23)	**Ne*	*hanno telefonato*	*molti.*
	of-them	called	many

(A.24)	*Li*	*ho*	*comprati.*
	them	have-1-S	bought-M-P
	I bought them		
(A.25)	*Maria*	*è*	*arrivata.*
	Maria	is-3-S	arrived-F-S
	Maria has arrived		
(A.26)	**Maria*	*ha*	*telefonata.*
	Maria	has-3-S	called-F-S.
	Maria has called (F)		

Unergatives There is some evidence that the inverted subjects of unergative verbs are adjoined to VP. The contrastive evidence with unaccusatives shows that the inverted subject is not in object position, on the other hand there is reason to think that the moved subject is inside the VP. For instance, moved subjects do not give rise to *that*-trace effects, showing that the position from which the *wh*-element moves is licensed by the ECP.

(A.27) a. Chi credi che verrà?
b. * Who do you think that will come?

The preverbal subject in this case is arguably *pro*, which would explain why null subjects, inverted subjects and lack of *that*-trace effects usually have been identified as a cluster of properties that depends on a language-dependent parameter, called the null subject parameter. This parameter establishes the restrictions on the distribution of the pronominal empty category *pro*.

This pleonastic subject is coindexed with the inverted lexical subject forming a chain. The reasons for this kind of coindexation are both theoretical and empirical, and they are drawn from Burzio (1986, 85ff). Firstly, by coindexing *pro* and the inverted subject, a parallel can be drawn with pleonastic chains, like *there* and *it* in English. In pleonastic chains there exist a relationship between the inverted NP and the pleonastic element

(A.28) a. There arrived a man.
b. It seems that John is here.

There only occurs with NPs, while *it* only occurs with sentences. Coindexation would explain this fact. Moreover, pleonastic chains show verb agreement. By coindexation of the inverted subject only one agreement mechanism would be necessary.

Chomsky (1981, 214) presents some empirical evidence for the coindexation of pleonastics. Given a condition for the interpretation of anaphors like A.29 and a restriction on coindexation like A.30, then the coindexation of an element with one of its constituents is ruled out.

(A.29) An anaphor is to be bound by its antecedent.
This is possible only if there is no closer subject or INFL
than the antecedent.

(A.30) *i-within-i* condition $*[_i \ldots a_i \ldots]$

Given these rules consider the following facts.

(A.31) a. * $They_i$ expect that [[each other $]_i$ will come].
b. $They_i$ think $[_{S1}$ it is a pity $[_{S2}$ that pictures of [each other $]_i$ are hanging on the wall.]]

The sentence in A.31a is incorrect because the closest binder for *each other* is not its antecedent but the INFL of *will come*. On the other hand, the fact that

A.31b is correct shows that the INFL of *are hanging* is not coindexed with *each other*. Under the standard assumption that verbs and subjects are coindexed, if the verb were coindexed with *each other* then the anaphor *each other* would also, transitively, be coindexed with *pictures of each other*, violating A.30. Hence *each other* is not bound inside S2. However, it is also not bound within S1, which means that *it* and S2 are coindexed.

We assume that Italian has a similar sort of chain as chains formed by pleonastic elements in English. In all these cases the notion of chain as a sequence of elements that share θ-role and Case is very important, because it permits to abstract away from the content of the θ-role and the direction of the assignment.

As Chomsky (1980a), Rizzi (1982) and Burzio (1986) all notice, coindexing *pro* and the inverted subject causes problems for the binding theory (see below), since *pro* binds the lexical NP. Several solutions have been suggested. Chomsky assumes that a different kind of indexing is involved here, that does not fall under the binding theory. Rizzi restricts the notion of *bound* element only to those elements that are not θ-dependent on their antecedents. Rizzi's solution fails to capture the similarity between inverted subjects of ergative verbs, which count as bound, and the other verbs. Burzio notes that Chomsky's solution fails to account for the locality restriction that apply to inverted subjects. Burzio suggest to reinterpret the conditions of Binding theory as meaning *argument bound* and *argument free*, so that in the case of an inverted subject, it would not violate binding theory because it would be bound by a non argument.

In conclusion, sentences with inverted subjects have different representations, depending on the base position of the *wh*-word, since *venire/verrà* (come) is unaccusative, while *telefonare/telefonerà* (call) is not. The chains in A.32 and A.33 represent quite clearly that the element t_i^k belongs to two chains: (pro^k, t_i^k) and (chi_i, t_i, t_i^k). The element t_i^k receives its Case and θ-role indirectly, through *pro* and then these features are transmitted to the $\overline{\text{A}}$ chain headed by *Chi*. Thus this chain differs from the other type of intersecting chain in two respects: first, the common element to the two chains is the rightmost lowermost element in the tree; second, the two chains share Case and θ-role, but they do not share coindexing, for the reasons related to binding theory.

(A.32) $[_{CP}$ chi$_i$ $[_{IP}$ credi [che pro$_k$ verrà$_l$ $[_{VP}$ t$_l$t$_i^k$]]]]

(A.33) $[_{CP}$ chi$_i$ $[_{IP}$ credi [che pro$_k$ *telefonerà*$_l$ $[_{VP}$ $[_{VP}$ t$_l$t$_i^k$]]]]]

A.2.6 Barriers Theory

Locality Restrictions

This is a collective name for all the restrictions that regulate the application of the movement rule move-α. They are listed them here under the headings of subjacency, relativized minimality, antecedent government and condition A of the binding theory.

Relativized Minimality (Rizzi 1990, 7)

X α-governs Y only if there is no such Z that
(i) Z is a typical potential α-governor for Y,
(ii) Z c-commands Y and does not c-command X.

Typical Potential Governor (Rizzi 1990, 7)

Z is a typical potential governor for Y iff
(i) if Y is in an A chain and Z is an A-specifier c-commanding Y
(ii) if Y is in an $\overline{A}$ chain and Z is an $\overline{A}$-specifier c-commanding Y
(iii) if Y is in a head chain and Z is a head c-commanding Y

A positions

A positions are positions to which, at least potentially, a θ-role could be assigned. In particular, the specifier of IP and NP are A positions.

$\overline{\text{A}}$ positions

$\overline{\text{A}}$ positions are all those positions that have not been listed as A positions. In particular, the Specifier of CP.

Barrier for Government (Cinque 1990, 42) (113)

Every maximal projection that fails to be directly selected by a category nondistinct from [+V] is a barrier for government.

Barrier for Binding (Cinque 1990, 42) (114)

Every maximal projection that fails to be (directly or indirectly) selected in the canonical direction by a category nondistinct from [+V] is a barrier for binding.

Subjacency

Long distance dependencies created by movement involve either material sitting in two adjacent clauses or material separated by an unbounded number of intervening clauses. It is never the case that items, let's say 4 clauses apart and only those, are related by a long-distance dependency. This fact can be captured quite elegantly by using rules that perform a basic step and an iterative step. The rule move-α which links elements such as *Who* and t_i can perform a simple basic step from a clause to an adjacent clause. Or it can iterate, and as a result, displace linguistic material unboundedly far away from the source position by a sequence of basic steps. For instance, in *Who do you think e′ that Mary saw e at the party? Who* is a displaced element that has been moved from the object position, after *saw*, here indicated by *e*. The well-formedness of such long-distance relations is regulated by the Subjacency Condition, which basically determines how big a single step can be.

In the formulation before Chomsky (1986a), Subjacency would hold in A.34b given A.34a.

(A.34) a. $\ldots \alpha \ldots [_\beta \ldots [_\delta \ldots \gamma]] \alpha \ldots$
b. No rule can relate α and γ in A.34a if two bounding nodes are intervening.

The result is that no single step in a cyclic application of movement could cross more than one bounding node. The bounding nodes are constants with the stipulated values of IP and NP for English. In more recent formulations

(Chomsky 1986a; Cinque 1990) bounding nodes coincide with barriers. (See presentation of linguistic facts in chapter 5.)

A.2.7 Trace Theory

The Empty Category Principle (ECP)

The Empty Category Principle (ECP) regulates the formal licensing of empty categories.

ECP (Cinque 1990, 49)

A nonpronominal empty category must be properly head governed by a head nondistinct form [+V].

Rizzi (1990) reformulates the Empty Category Principle (ECP), as a conjunction of conditions instead of the older disjunctive formulations. In previous versions, most notably Chomsky (1981),Chomsky (1986a) an empty category was licensed if it was *either* lexically governed *or* antecedent governed. Rizzi (1990) reformulates this condition as a conjunction: an empty category is properly governed if it is head governed *and* antecedent governed. The two conjuncts can be satisfied at different levels of representation. Namely, head government, which is a formal licensing principle, must be satisfied at S-structure. Antecedent government, on the other hand, can be satisfied at LF. In fact, antecedent government reduces to the binding of an anaphor that receives a referential θ-role, and as such it is subject to binding conditions. Thus the ECP is reduced to the condition above, which applies at S-structure.

Head Government (Rizzi 1990, 6)

X head-governs Y iff
(i) X $\in$ *{ A,N,V,P,Agr,T }*
(ii) X m-commands Y
(iii) no barrier intervenes
(iv) Relativized Minimality is respected.

Antecedent Government (Rizzi 1990, 6)

X antecedent-governs Y iff

(i) X and Y are coindexed

(ii) X c-commands Y

(iii) no barrier intervenes

(iv) Relativized Minimality is respected.

L Marking (Chomsky 1986a, 15)

α L-marks β iff α is a zero-level category that θ-marks β and α , β are sisters.

$\pm\gamma$

The feature $+\gamma$ is assigned to an empty category that has been formally licensed by the ECP.

A.2.8 Binding Theory

±anaphor

An NP bears the feature [+ anaphor] if it does not have independent reference, but it must be coindexed with another NP in the sentence to receive an interpretation. Thus, the trace of an NP is an anaphor, but also lexical items such as *himself, oneself, each other.*

±pronominal

An NP bears the feature [+ pronominal] if it can have both independent reference, and it can also be coindexed with another NP in the sentence to receive an interpretation. Thus, lexical items such as *him, she, their.*

Binding

The following principles, which constitute the core of binding theory, regulate the coreference of nominal expressions in a sentence.

Principle A (H91:216) *An anaphor must be A-bound in its governing category.*

Principle B (H91:216) *A pronominal must be A-free in its governing category.*

Principle C (H91:216) *An R-expression must be A-free.*

A-binding (H91:228)

α A-binds β iff
(i) α is in A-position
(ii) α c-commands β
(iii) α and β are coindexed

Governing Category (H91:229) *The Governing Category for α is the minimal domain containing α, its governor and an accessible SUBJECT.*

Accessible SUBJECT (H91:229) *α is an accessible SUBJECT for β if the coindexation of α and β does not violate any grammatical principle.*

Distribution of Empty Categories

According to the conditions of binding theory four types of referential objects can occur in natural languages, given by the combinatorics of the features [±anaphoric], [±pronominal]. In overt NPs we find that all the possible combinations of these features are attested, as shown below. In the best case the typology of empty categories should mirror the typology of overt NPs, with the

+anaphoric	−pronominal	*each other*	anaphor
−anaphoric	−pronominal	*John*	referential item
−anaphoric	+pronominal	*she*	pronouns
+anaphoric	+pronominal	impossible	no possible governing category hence Case filter violation

Table A.1
Typology of empty categories

difference that the last case of Table A.1 should be possible, since no Case need be assigned. And this is true, the last case being PRO.

Free Indexation

A basic assumption of the theory presented in Chomsky (1981),Chomsky (1982) about NPs, either lexical or nonlexical, is that they can be exhaustively partitioned by the features [±pronominal],[±anaphoric]. The argument has been used to justify the existence of *pro* and its features [+pronominal,-anaphoric]. (See null subjects below.) Chomsky (1982, 34) notices that trace and PRO are in (virtually) complementary distribution and that they (virtually) exhaustively cover the possible positions for NPs. Chomsky argues that this fact is explained if only one empty category is assumed, which is defined contextually. This is the so called *contextual* determination of empty categories: there is only one empty category that can take up different functions or occurrences in different contexts.

Brody (1984),Brody (1985) has shown that this argument is incorrect. Empirically, this interpretation would be supported by derivations where empty categories can change their status in the course of the derivation. The evidence for such derivations is unconvincing. Brody (1984, 360 fn 8) points out that the two typical cases are inverted subjects and parasitic gaps.

(A.35) t_i telefonano molte studentesse

In A.35 the trace created by rightward movement of the subject turns into a pronominal.

(A.36) Which book did you file t before reading e?

In A.36 the parasitic gap *e* must be a pronominal at D-structure, since it is free, but it becomes a variable at S-structure. Conceptually, as Brody (1984) points out, the assumption that there exist only one type of empty category does not entail the existence of contextual definitions. He argues that a random characterisation which is then filtered out by independently needed principles would also work. Brody (1984) is devoted to showing that contextual definitions are totally redundant and can be eliminated from the theory without any loss of empirical adequacy and with gain for the explanatoriness and economy of the theory. We give a sketch of Brody's line of argument. The contextual definitions for empty categories are the following (Chomsky 1981).

(A.37) 1. α is a pronominal iff
$\alpha = [_{NP}$ F,(P)], where P is a phonological matrix
and F $\subset \phi$ and either (i) or (ii).
(i) α is free
(ii) α is locally A-bound by a β with an independent
θ-role.

2. α is a variable iff α is locally $\overline{\text{A}}$-bound.

3. if α is an empty category and not a variable, then α
is an anaphor.

These definitions assign the correct features to the typology of empty categories, as can be seen from the table below. (LR means the from left to right and RL means from right to left).

Example	Features of e_x	Definition
Tom_x is illegal e_x to go there	nonpronominal	(A.371iiLR)
	anaphor	(A.373)
	nonvariable	(A.372RL)
Tom_x hit e_x	pronominal	(A.371iiRL)
	anaphor	(A.373)
	nonvariable	(A.372RL)
It_x is illegal e_x to go there	pronominal	(A.371iRL)
	anaphor	(A.373)
	nonvariable	(A.372RL)
Tom_x tried e_x to go there	pronominal	(A.371iiRL)
	anaphor	(A.373)
	nonvariable	(A.372RL)
Tom_x seems e_x to go there	nonpronominal	(A.371iiLR)
	anaphor	(A.373)
	nonvariable	(A.372RL)
It_x seems e_x to be obvious that S	nonpronominal	(A.371iiLR)
	anaphor	(A.373)
	nonvariable	(A.372RL)
Who_x did Tom hit e_x	nonpronominal	(A.371iiLR)
	variable	(A.372LR)

The different parts of the definitions are then shown to be reconducible to independently needed principles of the grammar.

(A.371iLR)	θ Criterion
(A.371iRL)	VEC, ECP, Principle A
(A.371iiLR)	θ Criterion
(A.371iiRL)	VEC, ECP, Principle C
(A.372LR)	VEC
(A.372RL)	VEC
(A.373)	identification principle, binding theory (A and B)

The VEC is the V-Element Condition, which states that a nonpronominal nonanaphor empty category must be bound while the identification principles is whatever property of clitics and Infl licenses *pro*.

Distribution of Null Subjects

I examine here the properties of null subjects. Firstly, one notices that such elements must exist because of the Extended Projection Principle, and that they undergo that same binding conditions as pronouns.

(A.38)	*Gianni/lui ∅*	*parla*	*sempre*	*di*	*se stesso.*
	Gianni/he	speaks	always	of	himself

" Gianni always talks about himself."

(A.39)	*Gianni/lui ∅*	*dice*	*che*	*Maria*	*parla*	*sempre*	*di*
	Gianni/he	says	that	Mary	speaks	always	of

*lui / *se stesso.*
him/* himself

" G. says that Mary always talks about him/ * himself"

Secondly, we need to establish the interpretation and distribution of the empty category that can be a null subject. According to Rizzi (1982, 130) (43)

(A.40) 1. A phonetically null subject with "dummy" interpretation can be found in the local context of a nominative assigner.

2. A phonetically null subject with definite pronominal interpretation can be found in the local context of a tensed inflection.

Some examples of null subjects in tensed clause are given in A.2.8 and null subjects in infinitival are given in A.2.8.

(A.41)	*pro*	*Verrà*
	"He'll	come"

(A.42)	*Credo*	*che*	*verrà*
	I think	he'll come	

(A.43) *Essendo piovuto per tutto il pomeriggio*
Having rained all afternoon,
non siamo usciti
we did not go out

(A.44) *Suppongo essere molto improbabile che Mario ci*
I-assume it to be very unlikely that Mario us
aiuti.
will help

Riguardo a F., avendo lei combinato questo pasticcio,
Regarding F., her having caused this mess,
sono nei guai fino al collo
I am in deep trouble

The null subjects of null subject languages have the same referential properties as pronouns, since they have independent reference and they can also be pleonastic. They differ from the other pronominal empty category PRO. Their distribution is, in fact, complementary to that of PRO in resumptive pronoun usage and weak cross over cases. PRO cannot be a resumptive pronouns, while *pro* can (Jaeggli and Safir 1989, 16).

(A.45) a. * That's the guy that we didn't know whether it was possible PRO to swim
b. * Questo è il tipo che non sapevamo se era possibile PRO nuotare

(A.46) a. That's the guy that we didn't know whether we should talk to him
b. Questo è il tipo che non sapevamo se fosse possibile parlargli

While A.46 shows that resumptive pronouns are licit both in Italian and English, A.47 shows that the null subject in the embedded clause is only licit in Italian, where it is *pro*. If the empty category in English were an NP trace it would violate Subjacency.

(A.47) a. That's the guy that Mary knows the woman whom he/* ∅ married.

b. Questo è il tipo che Maria conosce la donna che ?lui/ ∅ ha sposato.

Weak Cross Over is exhibited in those sentences, in which it is assumed that a single operator binds both a pronoun and a gap, as illustrated by A.48. In these configurations, PRO is licit, while *pro* is not, as shown in A.49 and A.50. Thus we can conclude that *pro* is [+pronominal,−anaphoric].

(A.48) * Who_i does his_i mother love t_i?

(A.49) Who_i did [PRO_i washing his_i car] upset t_i?

(A.50) * Chi_i accusò la donna con cui_j pro_i ballava t_j t_i ?

Finally, according to Jaeggli and Safir (1989). the null subject parameter is linked to the degree of morphological uniformity of the language.

(A.51) Null subjects are permitted in those languages with morphological uniform inflectional paradigm.

(A.52) An inflectional paradigm P in a language L is morphologically uniform iff P has either only underived inflectional forms or only derived inflectional forms.

Thus clearly Chinese allows null subjects because all forms are underived, while Italian or Spanish exhibit only derived inflectional forms. German, on the other hand, is not uniform.

A.3 PARSING ALGORITHMS

The design of the structure-building component of the parser is based on a shift-reduce method, the LR(k) method (Knuth 1965). LR(k) stands for Left-to-right rightmost derivation in Reverse, with k symbols of lookahead. I refer the reader to a standard text on parsing and compiling for a detailed description of the relevant concepts that will be used here: shift-reduce parsing, LR(k) parsing, LALR(k) parsing, for instance Aho and Ullman (1972).

Shift-Reduce

Shift-reduce parsing is a bottom up parsing method, which attempts to construct a parse tree for an input string starting at the leaves. It uses two main data structures: an input buffer and a stack.

Tokens of the input are *shifted* onto the stack until a substring of the tokens on top of the stack matches the right side of a rule of the grammar used by the parser. When such a substring is found, the substring is popped from the stack and substituted with the left side of the matching rule. In other words, the offspring nodes are substituted by their parent node. This process parses a sentence by tracing a rightmost derivation in reverse.[1]

For example, consider the grammar G in A.53.

(A.53)	1	IP $\rightarrow$NP I′
	2	I′ $\rightarrow$I0 VP
	3	VP $\rightarrow$V NP
	4	NP $\rightarrow$\{ John, Mary \}
	5	I0 $\rightarrow$\{ will \}
	6	V $\rightarrow$\{ help \}

Given G and the input string *John will help Mary*, a shift-reduce parser would perform the sequence of steps shown in Figure A.3, by consulting G when needed.

[1] A rightmost derivation is a derivation in which the rightmost nonterminal in each sentential form of the derivation is expanded first, *e.g.* S $\Rightarrow$NP VP $\Rightarrow$NP V NP $\Rightarrow$NP V *John* $\Rightarrow$NP *loves John* $\Rightarrow$*Mary loves John* is a rightmost derivation.

Action	Stack	Input
start	\$	John will help Mary
shift	\$ John	will help Mary
reduce by rule 4	\$ NP	will help Mary
shift	\$ NP will	help Mary
reduce by rule 5	\$ NP I0	help Mary
shift	\$ NP I0 help	Mary
reduce by rule 6	\$ NP I0 V	Mary
shift	\$ NP I0 V Mary	\$
reduce by rule 4	\$ NP I0 V NP	\$
reduce by rule 3	\$ NP I0 VP	\$
reduce by rule 2	\$ NP I′	\$
reduce by rule 1	\$ IP	\$
accept		

Figure A.3
Example of parse by the shift reduce method for the sentence *John will help Mary*. I use the symbol \$ to indicate the bottom of the stack and the end of the input string.

As can be noted, the parser can perform one of several actions at each step: it can *shift, reduce* or *accept*, or, if no other action is available, enter a state of *error*. Of course, the complex part of this procedure, which we have totally ignored so far, is how to recognize that the n tokens at the top of the stack correspond to a rule in the grammar, without performing an exhaustive search of all the sequences in the stack and all the rules in the grammar. The knowledge of when to shift or reduce and what to do next is contained in a look-up table, which is compiled off-line.

LR(k) Parsing

LR(k) is a deterministic version of shift-reduce parsing. An LR parser consists of an input, an output, an LR driver, a *goto* table and an *action* table. We show this schematically in Figure A.4, which is adapted from Aho and Ullman (1977, 217). As Aho and Ullman (1977, 215) point out, LR(k) parsing is attractive because the LR parsing method is the most general, non-backtracking shift-reduce parsing method known, yet it can be implemented as efficiently as other shift-reduce methods; moreover, an LR parser can detect an error as soon as it is possible to do so on a left-to-right scan of the input.

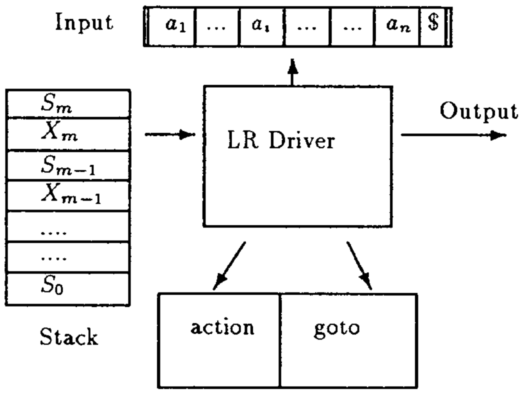


Figure A.4
Model of an LR Parser

The LR driver is the same for all parsers, while the *action* and *goto* tables change according to the grammar. The parsing program uses the state on top of the stack and the current input symbol as indices to consult the parse table. It determines the action to take, *e.g.* *shift*, and the next state.

A grammar is LR if it can be compiled into an LR table in such a way that each entry in the table contains a unique action and *goto* state. If a grammar is not LR, it is going to have conflicting actions. For example, the same table entry for such a grammar could contain a shift action and a reduce action (*shift/reduce conflict*), or it could contain reduce actions that point to different rules in the grammar (*reduce/reduce conflict*).

LALR(k)

LALR, which stands for LookaheadLR, is the most used method in practice, because it provides the same performance advantages as the canonical LR method, but it requires tables that are much smaller. For instance, for a typical programming language like Pascal, an LALR parse table has hundreds of states, while an LR parser table has thousands of states. The reduction in size of the state set is the result of a "collapsing" mechanism. For example, an LR(k) parser with $k = 1$ could have the following two states in its canonical collection of items.

(A.54) $I_1 = \{\mathrm{L} \rightarrow \mathbf{id} \bullet, \$, = \}$
$I_2 = \{\mathrm{L} \rightarrow \mathbf{id} \bullet, \$ \}$

The two items mean that **id** must be reduced to L, *i.e.* **id** has been recognized as a token of category L, if it is the last input token ($) or if it is followed by the token =. I_2 is an extensional subset of I_1. This means that there is a set of inputs for which the two states are equivalent. States that have this kind of relation are said to have the same *core*, and they are reduced to one state. The only consequence of this collapsing procedure is that, while an LR(k) parser fails as soon as an error is encountered, an equivalent LALR(k) parser might perform some unnecessary shift actions before failing. The LALR(k) will never reduce incorrectly, though. For proofs of strong equivalence of the two methods and a more detailed explanation see Aho and Ullman (1972), and Aho and Ullman (1977).

B
RESULTS

In this appendix I show snapshots of the parses of several grammatical constructions. In the interest of space, I present only the most relevant steps in the parse, while the other states are simply shown by showing the state number. It is to be understood that the output has been formatted. The states in the parse that are shown in full have not been modified.

This subset of constructions has been chosen as it constitutes the crucial test set for modular parsers, since it involves complex interactions of modules over large portions of the tree to compute long distance dependencies. Many other proposals either do not deal with all types of chains (Frank 1992; Johnson 1989), for instance, or they require extensive backtracking (Fong 1991; Fong and Berwick 1992).

Simple Transitive

```
INPUT TOKENS: john  loves  mary

state : 0
state : 2
-----------------------------------------------
state : 5
token : w0
stack:  (john,3,m,s(n,[],[bar(2),proper]),fts(_,_))
Abar chains:
A chains:
 (john,0,m,s(n,[],[bar(2),proper]),fts(_,_))
-----------------------------------------------
```

```
state : 6
state : 1
-----------------------------------------------
state : 6
token : w2
stack:  (love+s,6,h,s(v,[(_3551,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,4,h,s(infl,[],[empty]),fts(_,_))
        (john,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
 (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 2
-----------------------------------------------
state : 9
token : end_of_file
stack:  (mary,7,m,s(n,[],[bar(2),proper]),fts(_,_))
        (love+s,6,h,s(v,[(_3551,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,4,h,s(infl,[],[empty]),fts(_,_))
        (john,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (mary,2,m,s(n,[],[bar(2),proper]),fts(_,_))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 4
token : end_of_file
stack:  (_,8,p,s(v,[(d,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,4,h,s(infl,[],[empty]),fts(_,_))
        (john,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (mary,2,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 8
state : 9
state : 7
state : 3
state : 10

accepted
```

```
PARSE TREE :
                                 infl2
                                   |
                         .------------.
                         |            |
                        n2          infl1
                         |            |
                         .     .----------.
                         |     |          |
                        n2   infl0       v1
                         |     |          |
                         .     .       .------.
                         |     |       |      |
                       john    e      v0     n2
                                       |      |
                                       .      .
                                       |      |
                                      v0     n2
                                       |      |
                                       .      .
                                       |      |
                                    love+s  mary
Abar chains:
A chains:
        (mary,2,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
```

Simple Intransitive

```
INPUT TOKENS: john  runs
state : 0
state : 2
-------------------------------------------------
state : 5
stack:  (john,2,m,s(n,[],[bar(2),proper]),fts(_,_))
Abar chains:
A chains:
 (john,0,m,s(n,[],[bar(2),proper]),fts(_,_))
-------------------------------------------------
state : 6
state : 1
-------------------------------------------------
state : 6
token : end_of_file
stack:  (run+s,5,h,s(v,[(d,ext,ag)],[]),fts(_,_))
        (_,3,h,s(infl,[],[empty]),fts(_,_))
        (john,2,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
 (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-------------------------------------------------
state : 4
state : 8
state : 9
state : 7
state : 3
state : 10
accepted

PARSE TREE :                        infl2
                                      |
                               .----------.
                               |          |
                              n2        infl1
                               |          |
                               .      .------.
                               |      |      |
                              n2    infl0   v0
                               |      |      |
                               .      .      .
                               |      |      |
                             john     e     v0
                                             |
                                             .
                                             |
                                           run+s
Abar chains:
A chains:
 (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
```

Simple Passive

```
INPUT TOKENS: mary  was  loved

state : 0
state : 2
-----------------------------------------------
state : 5
token : w0
stack:  (mary,3,m,s(n,[],[bar(2),proper]),fts(_,_))
Abar chains:
A chains:
 (mary,0,m,s(n,[],[bar(2),proper]),fts(_,_))
-----------------------------------------------
state : 1
state : 6
state : 1
-----------------------------------------------
state : 6
token : end_of_file
stack:  (loved,6,h,s(v,[(_,nil,th),(d,ext,nil)],[pass]),fts(_,_))
        (was,5,h,s(infl,[(_,nil,pred),(_,ext,_)],[aux(s)]),fts(_,_))
        (mary,3,m,s(n,[],[bar(2),proper]),fts(nil,ext))
Abar chains:
A chains:
 (mary,0,m,s(n,[],[bar(2),proper]),fts(nil,ext))
-----------------------------------------------
state : 9
token : end_of_file
stack:  (_,8,m,s(n,[],[empty]),fts(_,_))
        (loved,6,h,s(v,[(_,nil,th),(d,ext,nil)],[pass]),fts(_,_))
        (was,5,h,s(infl,[(_,nil,pred),(_,ext,_)],[aux(s)]),fts(_,_))
       7
        (mary,3,m,s(n,[],[bar(2),proper]),fts(nil,ext))
Abar chains:
A chains:
 (mary,0,m,s(n,[],[bar(2),proper]),fts(nil,ext))
-----------------------------------------------
state : 4
token : end_of_file
stack:  (_,9,p,s(v,[(d,nil,th),(d,ext,nil)],[pass]),fts(_,_))
        (was,5,h,s(infl,[(_,nil,pred),(_,ext,_)],[aux(s)]),fts(_,_))
       7
        (mary,3,m,s(n,[],[bar(2),proper]),fts(nil,ext))
Abar chains:
A chains:
 (mary,[0|8],_,s(n,[],[bar(2),proper]),fts(th,ext))
-----------------------------------------------
state : 8
state : 9
state : 7
state : 3
state : 10

accepted

PARSE TREE :                      infl2
                                    |
                           .-----------.
                           |           |
                          n2         infl1
                           |           |
                           .     .--------.
                           |     |        |
                          n2   infl0     v1
                           |     |        |
                           .     .      .-----.
                           |     |      |     |
                         mary  infl0   v0    n2
                                 |      |     |
                                 .      .     .
                                 |      |     |
                                was    v0     e
                                        |
                                        .
                                        |
                                      loved

Abar chains:
A chains:
 (mary,[0|8],_,s(n,[],[bar(2),proper]),fts(th,ext))
```

Simple Raising

```
INPUT TOKENS: mary  seems  to  like  john  .

state : 0
state : 2
state : 5
state : 6
-----------------------------------------------
state : 1
token : w0
stack:  (seem+s,1,h,s(v,[(_3867,nil,th),(_,ext,nil)],[rais]),fts(_,_))
        (_,6,h,s(infl,[],[empty]),fts(_,_))
        (mary,5,m,s(n,[],[bar(2),proper]),fts(_,_))
Abar chains:
A chains:
 (mary,0,m,s(n,[],[bar(2),proper]),fts(_,_))
-----------------------------------------------
state : 6
token : w0
stack:  (seem+s,8,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,6,h,s(infl,[],[empty]),fts(_,_))
        (mary,5,m,s(n,[],[bar(2),proper]),fts(nil,ext))
Abar chains:
A chains:
 (mary,0,m,s(n,[],[bar(2),proper]),fts(nil,ext))
-----------------------------------------------
state : 9
state : 1
state : 6
state : 1
state : 6
state : 2
-----------------------------------------------
state : 9
token : end_of_file
stack:  (john,15,m,s(n,[],[bar(2),proper]),fts(_,_))
        (like,14,h,s(v,[(_,acc,th),(_,ext,ag)],[]),fts(_,_))
        (to,12,h,s(infl,v,[inf]),fts(_,_))
        (_,10,m,s(n,[],[empty]),fts(ag,nil))
        (seem+s,8,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,6,h,s(infl,[],[empty]),fts(_,_))
       9
        (mary,5,m,s(n,[],[bar(2),proper]),fts(nil,ext))
Abar chains:
A chains:
        (john,4,m,s(n,[],[bar(2),proper]),fts(_,_))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(nil,ext))
-----------------------------------------------
state : 4
token : end_of_file
stack:  (_,16,p,s(v,[(d,acc,th),(_5627,ext,ag)],[]),fts(_,_))
```

```
        (to,12,h,s(infl,v,[inf]),fts(_,_))
        (_,10,m,s(n,[],[empty]),fts(ag,nil))
        (seem+s,8,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,6,h,s(infl,[],[empty]),fts(_,_))
       9
        (mary,5,m,s(n,[],[bar(2),proper]),fts(nil,ext))
Abar chains:
A chains:
        (john,4,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(nil,ext))
-----------------------------------------------
state : 8
state : 9
state : 7
-----------------------------------------------
state : 8
token : end_of_file
stack:  (_,20,m,s(infl,v,[inf]),fts(_,_))
        (seem+s,8,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,6,h,s(infl,[],[empty]),fts(_,_))
       9
        (mary,5,m,s(n,[],[bar(2),proper]),fts(nil,ext))
Abar chains:
A chains:
        (john,4,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (mary,[0|10],_,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 9
state : 4
state : 8
state : 9
state : 7
state : 3
state : 10
accepted
```

```
PARSE TREE :                       infl2
                                     |
                  .----------------------.
                  |                      |
                 n2                   infl1
                  |                      |
                  .     .-------------------.
                  |     |                   |
                 n2   infl0               v1
                  |     |                   |
                  .     .       .---------------.
                  |     |       |               |
                mary    e      v0            infl2
                                |               |
                                .      .----------.
                                |      |          |
                               v0    n2        infl1
                                |      |          |
                                .      .    .---------.
                                |      |    |         |
                             seem+s    e infl0      v1
                                            |         |
                                            .      .-----.
                                            |      |     |
                                         infl0    v0    n2
                                            |      |     |
                                            .      .     .
                                            |      |     |
                                           to     v0    n2
                                                   |     |
                                                like   john
```

```
Abar chains:
A chains:
        (john,4,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (mary,[0|10],_,s(n,[],[bar(2),proper]),fts(ag,ext))
```

Embedded Transitive

```
INPUT TOKENS: john  thinks  that  mary  loves  bill  .

state : 0
state : 2
state : 5
state : 6
------------------------------------------------
state : 1
token : w0
stack:  (think+s,1,h,s(v,[(_,prop,th),(_,ext,ag)],[]),fts(_,_))
        (_,7,h,s(infl,[],[empty]),fts(_,_))
        (john,6,m,s(n,[],[bar(2),proper]),fts(_,_))
Abar chains:
A chains:
 (john,0,m,s(n,[],[bar(2),proper]),fts(_,_))
------------------------------------------------
state : 6
token : w0
stack:  (think+s,9,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,7,h,s(infl,[],[empty]),fts(_,_))
        (john,6,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
 (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
------------------------------------------------
state : 1
state : 6
state : 2
state : 9
------------------------------------------------
state : 6
token : w0
stack:  (_,15,h,s(infl,[],[empty]),fts(_,_))
        (mary,14,m,s(n,[],[bar(2),proper]),fts(_,_))
        (that,13,h,s(comp,infl,[]),fts(_,_))
        (think+s,9,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,7,h,s(infl,[],[empty]),fts(_,_))
        (john,6,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (mary,3,m,s(n,[],[bar(2),proper]),fts(_,_))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
------------------------------------------------
state : 1
------------------------------------------------
state : 6
token : w2
stack:  (love+s,17,h,s(v,[(_,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,15,h,s(infl,[],[empty]),fts(_,_))
        (mary,14,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (that,13,h,s(comp,infl,[]),fts(_,_))
        (think+s,9,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,7,h,s(infl,[],[empty]),fts(_,_))
        (john,6,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (mary,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 2
-----------------------------------------------
state : 9
token : end_of_file
stack:  (bill,18,m,s(n,[],[bar(2),proper]),fts(_,_))
        (love+s,17,h,s(v,[(_,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,15,h,s(infl,[],[empty]),fts(_,_))
        (mary,14,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (that,13,h,s(comp,infl,[]),fts(_,_))
        (think+s,9,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,7,h,s(infl,[],[empty]),fts(_,_))
        (john,6,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (bill,5,m,s(n,[],[bar(2),proper]),fts(_,_))
        (mary,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 4
token : end_of_file
stack:  (_,19,p,s(v,[(d,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,15,h,s(infl,[],[empty]),fts(_,_))
        (mary,14,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (that,13,h,s(comp,infl,[]),fts(_,_))
        (think+s,9,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,7,h,s(infl,[],[empty]),fts(_,_))
        (john,6,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (bill,5,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (mary,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 8
state : 9
state : 7
state : 8
state : 9
state : 4
state : 8
state : 9
state : 4
state : 8
state : 9
state : 7
state : 3
state : 10

accepted


PARSE TREE :
                                  infl2
                                    |
           .---------------------------.
           |                           |
          n2                         infl1
           |                           |
           .     .------------------------.
           |     |                        |
          n2   infl0                     v1
           |     |                        |
           .     .       .---------------------.
           |     |       |                     |
         john    e      v0                   comp1
                         |                     |
                         .       .----------------.
                         |       |                |
                        v0     comp0            infl2
                         |       |                |
                         .       .      .------------.
                         |       |      |            |
                      think+s  comp0   n2          infl1
                                 |      |            |
                                 .      .     .----------.
                                 |      |     |          |
                               that    n2   infl0       v1
                                        |     |          |
                                        .     .       .------.
                                        |     |       |      |
                                      mary    e      v0     n2
                                                      |      |
                                                      .      .
                                                      |      |
                                                     v0     n2
                                                      |      |
                                                      .      .
                                                      |      |
                                                   love+s  bill

Abar chains:
A chains:
        (bill,5,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (mary,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
```

Embedded Intransitive

This construction behaves analogously to the previous one, as far as feature annotation is concerned. Therefore, I show only the final output, with the correctly annotated chains.

```
INPUT TOKENS: john  thinks  that  mary  runs  .

------------------------------------------------
accepted

PARSE TREE:                      infl2
                                   |
              .-----------------------.
              |                       |
             n2                     infl1
              |                       |
              .     .---------------------.
              |     |                     |
             n2   infl0                  v1
              |     |                     |
              .     .       .-----------------.
              |     |       |                 |
            john    e      v0               compl
                            |                 |
                            .       .-------------.
                            |       |             |
                           v0     comp0         infl2
                            |       |             |
                            .       .      .---------.
                            |       |      |         |
                         think+s  comp0   n2       infl1
                                    |      |         |
                                    .      .     .------.
                                    |      |     |      |
                                  that    n2   infl0   v0
                                           |     |      |
                                           .     .      .
                                           |     |      |
                                         mary    e     v0
                                                        |
                                                        .
                                                        |
                                                      run+s

Abar chains:
A chains:
        (mary,3,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (john,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
```

Embedded Raising

```
INPUT TOKENS: mary  thinks  that  john  seems  to  like  bill  .

state : 0
state : 2
-----------------------------------------------
state : 5
token : w0
stack:  (mary,8,m,s(n,[],[bar(2),proper]),fts(_,_))
Abar chains:
A chains:
 (mary,0,m,s(n,[],[bar(2),proper]),fts(_,_))
-----------------------------------------------
state : 6
state : 1
-----------------------------------------------
state : 6
token : w0
stack:  (think+s,11,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,9,h,s(infl,[],[empty]),fts(_,_))
        (mary,8,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
 (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 1
state : 6
state : 2
-----------------------------------------------
state : 9
token : w0
stack:  (john,16,m,s(n,[],[bar(2),proper]),fts(_,_))
        (that,15,h,s(comp,infl,[]),fts(_,_))
        (think+s,11,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,9,h,s(infl,[],[empty]),fts(_,_))
        (mary,8,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (john,3,m,s(n,[],[bar(2),proper]),fts(_,_))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 6
state : 1
-----------------------------------------------
state : 6
token : w0
stack:  (seem+s,19,h,s(v,[(_8441,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,17,h,s(infl,[],[empty]),fts(_,_))
        (john,16,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,15,h,s(comp,infl,[]),fts(_,_))
        (think+s,11,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,9,h,s(infl,[],[empty]),fts(_,_))
        (mary,8,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (john,3,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-------------------------------------------------
state : 9
token : w0
stack:  (_,21,m,s(n,[],[empty]),fts(_,_))
        (seem+s,19,h,s(v,[(_8441,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,17,h,s(infl,[],[empty]),fts(_,_))
       20
        (john,16,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,15,h,s(comp,infl,[]),fts(_,_))
        (think+s,11,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,9,h,s(infl,[],[empty]),fts(_,_))
        (mary,8,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (john,3,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-------------------------------------------------
state : 1
state : 6
state : 1
state : 6
state : 2
-------------------------------------------------
state : 9
token : end_of_file
stack:  (bill,26,m,s(n,[],[bar(2),proper]),fts(_,_))
        (like,25,h,s(v,[(_10189,acc,th),(_10201,ext,ag)],[]),fts(_,_))
        (to,23,h,s(infl,v,[inf]),fts(_,_))
        (_,21,m,s(n,[],[empty]),fts(ag,nil))
        (seem+s,19,h,s(v,[(_8441,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,17,h,s(infl,[],[empty]),fts(_,_))
       20
        (john,16,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,15,h,s(comp,infl,[]),fts(_,_))
        (think+s,11,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,9,h,s(infl,[],[empty]),fts(_,_))
        (mary,8,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (bill,7,m,s(n,[],[bar(2),proper]),fts(_,_))
        (john,3,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-------------------------------------------------
state : 4
token : end_of_file
stack:  (_,27,p,s(v,[(d,acc,th),(_10201,ext,ag)],[]),fts(_,_))
```

```
        (to,23,h,s(infl,v,[inf]),fts(_,_))
        (_,21,m,s(n,[],[empty]),fts(ag,nil))
        (seem+s,19,h,s(v,[(_8441,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,17,h,s(infl,[],[empty]),fts(_,_))
       20
        (john,16,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,15,h,s(comp,infl,[]),fts(_,_))
        (think+s,11,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,9,h,s(infl,[],[empty]),fts(_,_))
        (mary,8,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (bill,7,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (john,3,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 8
state : 9
state : 7
-----------------------------------------------
state : 8
token : end_of_file
stack:  (_,31,m,s(infl,v,[inf]),fts(_,_))
        (seem+s,19,h,s(v,[(_8441,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,17,h,s(infl,[],[empty]),fts(_,_))
       20
        (john,16,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,15,h,s(comp,infl,[]),fts(_,_))
        (think+s,11,h,s(v,[(_,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,9,h,s(infl,[],[empty]),fts(_,_))
        (mary,8,m,s(n,[],[bar(2),proper]),fts(ag,ext))
Abar chains:
A chains:
        (bill,7,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (john,[3|21],_,s(n,[],[bar(2),proper]),fts(ag,ext))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-----------------------------------------------
state : 9
state : 4
state : 8
state : 9
state : 7
state : 8
state : 9
state : 4
state : 8
state : 9
state : 4
state : 8
state : 9
state : 7
state : 3
state : 10
accepted

PARSE TREE :                     infl2
                                   |
  .-----------------------------------.
 n2                                 infl1
  |                                   |
  .     .---------------------------------.
 n2   infl0                              v1
  |     |                                 |
  .     .       .-----------------------------.
mary    e      v0                           comp1
                |                             |
                .       .-------------------------.
               v0     comp0                     infl2
                |       |                         |
                .       .      .---------------------.
             think+s  comp0   n2                   infl1
                        |      |                     |
                        .      .     .------------------.
                      that    n2   infl0               v1
                               |     |                  |
                               .     .       .--------------.
                             john    e      v0            infl2
                                             |              |
                                             .     .----------.
                                            v0    n2        infl1
                                             |     |          |
                                             .     .    .---------.
                                          seem+s   e  infl0      v1
                                                        |         |
                                                        .      .-----.
                                                      infl0   v0    n2
                                                        |      |     |
                                                        .      .     .
                                                       to     v0    n2
                                                               |     |
                                                             like  bill
Abar chains:
A chains:
        (bill,7,m,s(n,[],[bar(2),proper]),fts(th,acc))
        (john,[3|21],_,s(n,[],[bar(2),proper]),fts(ag,ext))
        (mary,0,m,s(n,[],[bar(2),proper]),fts(ag,ext))
```

Simple Question

```
INPUT TOKENS: who  does  john  love  ?

state : 0
state : 2
-------------------------------------------------
state : 5
token : w0
stack:  (who,4,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
-------------------------------------------------
state : 1
state : 6
state : 2
-------------------------------------------------
state : 9
token : w0
stack:  (john,7,m,s(n,[],[bar(2),proper]),fts(_,_))
        (does,6,h,s(comp,infl,[]),fts(_,_))
        (who,4,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
 (john,2,m,s(n,[],[bar(2),proper]),fts(_,_))
-------------------------------------------------
state : 6
state : 1
state : 6
-------------------------------------------------
state : 9
token : end_of_file
stack:  (_,12,m,s(n,[],[empty]),fts(_,_))
        (love,10,h,s(v,[(_4750,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,8,h,s(infl,[],[empty]),fts(_,_))
        (john,7,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (does,6,h,s(comp,infl,[]),fts(_,_))
       11
        (who,4,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
 (john,2,m,s(n,[],[bar(2),proper]),fts(ag,ext))
-------------------------------------------------
state : 4
token : end_of_file
stack:  (_,13,p,s(v,[(d,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,8,h,s(infl,[],[empty]),fts(_,_))
        (john,7,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (does,6,h,s(comp,infl,[]),fts(_,_))
       11
        (who,4,m,s(n,[],[bar(2),wh]),fts(nil,nil))
Abar chains:
 (who,[0|12],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
 (john,2,m,s(n,[],[bar(2),proper]),fts(ag,ext))
------------------------------------------------
state : 8
state : 9
state : 7
state : 8
state : 9
state : 7
state : 3
state : 10
accepted
PARSE TREE :
                                 comp2
                                   |
                    .-----------------.
                    |                 |
                   n2               comp1
                    |                 |
                    .     .--------------.
                    |     |              |
                   n2   comp0          infl2
                    |     |              |
                    .     .      .----------.
                    |     |      |          |
                   who  comp0   n2        infl1
                          |      |          |
                          .      .     .--------.
                          |      |     |        |
                        does    n2   infl0     v1
                                 |     |        |
                                 .     .      .----.
                                 |     |      |    |
                               john    e     v0   n2
                                              |    |
                                              .    .
                                              |    |
                                             v0    e
                                              |
                                            love
Abar chains:
 (who,[0|12],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
 (john,2,m,s(n,[],[bar(2),proper]),fts(ag,ext))
```

Embedded Question

```
INPUT TOKENS: who  do  you  think  that  john  likes  ?

                                 comp2
                                   |
    .---------------------------------.
   n2                               comp1
    |                                 |
    .     .------------------------------.
   n2   comp0                          infl2
    |     |                              |
    .     .     .---------------------------.
   who  comp0  n2                         infl1
          |     |                           |
          .     .     .------------------------.
         do    n2   infl0                     v1
                |     |                        |
                .     .      .---------------------.
               you    e     v0                  comp2
                             |                     |
                             .     .-----------------.
                            v0   n2               comp1
                             |     |                 |
                             .     .    .---------------.
                           think   e comp0           infl2
                                        |               |
                                        .      .-----------.
                                     comp0   n2         infl1
                                        |      |           |
                                        .      .     .---------.
                                     that    n2   infl0      v1
                                               |     |         |
                                               .     .       .-----.
                                            john    e      v0    n2
                                                             |     |
                                                           v0     e
                                                             |
                                                         like+s
Abar chains:
         (who,[[0|23]|15],_,s(n,[],[bar(2),wh]),fts(th,acc))

A chains:
        (john,5,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
```

Embedded Question and Raising

```
INPUT TOKENS: who  did  you  think  that  john  seemed  to  like  ?

state : 0
state : 2
state : 5
state : 1
state : 6
state : 2
-----------------------------------------------
state : 9
token : w0
stack:  (you,12,m,s(n,[],[bar(2)]),fts(_,_))
        (did,11,h,s(comp,infl,[]),fts(_,_))
        (who,9,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
 (you,2,m,s(n,[],[bar(2)]),fts(_,_))
-----------------------------------------------
state : 6
state : 1
-----------------------------------------------
state : 6
token : w0
stack:  (think,15,h,s(v,[(_5854,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,13,h,s(infl,[],[empty]),fts(_,_))
        (you,12,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,11,h,s(comp,infl,[]),fts(_,_))
        (who,9,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
 (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 9
state : 1
state : 6
-----------------------------------------------
state : 2
token : w0
stack:  (john,5,m,s(n,[],[bar(2),proper]),fts(_,_))
        (that,19,h,s(comp,infl,[]),fts(_,_))
        (_,17,m,s(n,[],[empty]),fts(_,_))
        (think,15,h,s(v,[(_5854,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,13,h,s(infl,[],[empty]),fts(_,_))
        (you,12,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,11,h,s(comp,infl,[]),fts(_,_))
       16
        (who,9,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
 (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 9
-----------------------------------------------
state : 6
token : w0
stack:  (_,21,h,s(infl,[],[empty]),fts(_,_))
        (john,20,m,s(n,[],[bar(2),proper]),fts(_,_))
        (that,19,h,s(comp,infl,[]),fts(_,_))
        (_,17,m,s(n,[],[empty]),fts(_,_))
        (think,15,h,s(v,[(_5854,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,13,h,s(infl,[],[empty]),fts(_,_))
        (you,12,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,11,h,s(comp,infl,[]),fts(_,_))
       16
        (who,9,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
        (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
        (john,5,m,s(n,[],[bar(2),proper]),fts(_,_))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 1
-----------------------------------------------
state : 6
token : w0
stack:  (seem+ed,23,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,21,h,s(infl,[],[empty]),fts(_,_))
        (john,20,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,19,h,s(comp,infl,[]),fts(_,_))
        (_,17,m,s(n,[],[empty]),fts(_,_))
        (think,15,h,s(v,[(_5854,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,13,h,s(infl,[],[empty]),fts(_,_))
        (you,12,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,11,h,s(comp,infl,[]),fts(_,_))
       16
        (who,9,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
        (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
        (john,5,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 9
state : 1
state : 6
state : 1
state : 6
state : 9
-----------------------------------------------
```

```
state : 4
token : end_of_file
stack:  (_,32,p,s(v,[(d,acc,th),(_12860,ext,ag)],[]),fts(_,_))
        (to,27,h,s(infl,v,[inf]),fts(_,_))
       30
        (who,25,m,s(n,[],[empty]),fts(ag,nil))
        (seem+ed,23,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,21,h,s(infl,[],[empty]),fts(_,_))
       24
        (john,20,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,19,h,s(comp,infl,[]),fts(_,_))
        (_,17,m,s(n,[],[empty]),fts(_,_))
        (think,15,h,s(v,[(_5854,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,13,h,s(infl,[],[empty]),fts(_,_))
        (you,12,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,11,h,s(comp,infl,[]),fts(_,_))
       16
        (who,9,m,s(n,[],[bar(2),wh]),fts(nil,nil))
Abar chains:
        (who,[0|31],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
        (john,5,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))


-----------------------------------------------
state : 8
token : end_of_file
stack:  (_,33,m,s(v,[(d,acc,th),(_12860,ext,ag)],[]),fts(_,_))
        (to,27,h,s(infl,v,[inf]),fts(_,_))
       30
        (who,25,m,s(n,[],[empty]),fts(ag,nil))
        (seem+ed,23,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,21,h,s(infl,[],[empty]),fts(_,_))
       24
        (john,20,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,19,h,s(comp,infl,[]),fts(_,_))
        (_,17,m,s(n,[],[empty]),fts(_,_))
        (think,15,h,s(v,[(_5854,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,13,h,s(infl,[],[empty]),fts(_,_))
        (you,12,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,11,h,s(comp,infl,[]),fts(_,_))
       16
        (who,9,m,s(n,[],[bar(2),wh]),fts(nil,nil))
Abar chains:
        (who,[0|31],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
        (john,5,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 9
state : 7
-----------------------------------------------
```

```
state : 8
token : end_of_file
stack:  (_,36,m,s(infl,v,[inf]),fts(_,_))
        (seem+ed,23,h,s(v,[(_,nil,th),(d,ext,nil)],[rais]),fts(_,_))
        (_,21,h,s(infl,[],[empty]),fts(_,_))
       24
        (john,20,m,s(n,[],[bar(2),proper]),fts(nil,ext))
        (that,19,h,s(comp,infl,[]),fts(_,_))
        (_,17,m,s(n,[],[empty]),fts(_,_))
        (think,15,h,s(v,[(_5854,prop,th),(d,ext,ag)],[]),fts(_,_))
        (_,13,h,s(infl,[],[empty]),fts(_,_))
        (you,12,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,11,h,s(comp,infl,[]),fts(_,_))
       16
        (who,9,m,s(n,[],[bar(2),wh]),fts(nil,nil))
Abar chains:
 (who,[0|31],_,s(n,[],[bar(2),wh]),fts(th,acc))
A chains:
        (john,[5|25],_,s(n,[],[bar(2),proper]),fts(ag,ext))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 9
state : 4
state : 8
state : 9
state : 7
state : 8
state : 9
state : 7
state : 8
state : 9
state : 4
state : 8
state : 9
state : 7
state : 8
state : 9
state : 7
state : 3
state : 10
-----------------------------------------------
accepted
```

```
PARSE TREE :                           comp2
                                         |
 .------------------------------------------.
n2                                        comp1
 |                                          |
 .     .---------------------------------------.
n2   comp0                                   infl2
 |     |                                       |
 .     .     .------------------------------------.
who  comp0  n2                                  infl1
       |     |                                    |
       .     .     .---------------------------------.
      did   n2   infl0                              v1
             |     |                                 |
             .     .       .-----------------------------.
            you    e     v0                            comp2
                          |                              |
                          .      .--------------------------.
                         v0    n2                        comp1
                          |      |                         |
                          .      .    .------------------------.
                        think    e  comp0                   infl2
                                      |                       |
                                      .     .--------------------.
                                   comp0   n2                  infl1
                                      |     |                    |
                                      .     .     .------------------.
                                   that    n2   infl0              v1
                                            |     |                  |
                                            .     .        .-------------.
                                          john    e      v0            infl2
                                                          |              |
                                                          .       .--------.
                                                         v0     n2       infl1
                                                          |       |        |
                                                          .       .   .--------.
                                                       seem+ed    e  infl0    v1
                                                                       |       |
                                                                       .      .----.
                                                                     infl0  v0   n2
                                                                       |      |   |
                                                                       .      .   .
                                                                     to     v0    e
                                                                             |
                                                                            like
 Abar chains:
         (who,[[0|31]|17],_,s(n,[],[bar(2),wh]),fts(th,acc))
 A chains:
         (john,[5|25],_,s(n,[],[bar(2),proper]),fts(ag,ext))
         (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
```

Embedded Wh-Question

```
INPUT TOKENS: who  did  you  wonder  why  mary  liked  ?

state : 0
state : 2
-----------------------------------------------
state : 5
token : w0
stack:  (who,7,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
-----------------------------------------------
state : 1
state : 6
state : 2
-----------------------------------------------
state : 9
token : w0
stack:  (you,10,m,s(n,[],[bar(2)]),fts(_,_))
        (did,9,h,s(comp,infl,[]),fts(_,_))
        (who,7,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
 (you,2,m,s(n,[],[bar(2)]),fts(_,_))
-----------------------------------------------
state : 6
state : 1
state : 6
-----------------------------------------------
state : 2
token : w2
stack:  (why,4,m,s(adv,[],[bar(2),wh]),fts(_,_))
        (wonder,13,h,s(v,[(_5666,prop,nil),(d,ext,ag)],[]),fts(_,_))
        (_,11,h,s(infl,[],[empty]),fts(_,_))
        (you,10,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,9,h,s(comp,infl,[]),fts(_,_))
        (who,7,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
 (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 9
state : 6
state : 2
state : 9
-----------------------------------------------
state : 6
token : w0
```

```
stack:  (_,17,h,s(infl,[],[empty]),fts(_,_))
        (mary,16,m,s(n,[],[bar(2),proper]),fts(_,_))
        (_,15,h,s(comp,[],[empty]),fts(_,_))
        (why,14,m,s(adv,[],[bar(2),wh]),fts(_,_))
        (wonder,13,h,s(v,[(_5666,prop,nil),(d,ext,ag)],[]),fts(_,_))
        (_,11,h,s(infl,[],[empty]),fts(_,_))
        (you,10,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,9,h,s(comp,infl,[]),fts(_,_))
        (who,7,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
        (mary,5,m,s(n,[],[bar(2),proper]),fts(_,_))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))
-----------------------------------------------
state : 1
-----------------------------------------------
state : 6
token : end_of_file
stack:  (like+d,19,h,s(v,[(_9314,acc,th),(d,ext,ag)],[]),fts(_,_))
        (_,17,h,s(infl,[],[empty]),fts(_,_))
        (mary,16,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (_,15,h,s(comp,[],[empty]),fts(_,_))
        (why,14,m,s(adv,[],[bar(2),wh]),fts(_,_))
        (wonder,13,h,s(v,[(_5666,prop,nil),(d,ext,ag)],[]),fts(_,_))
        (_,11,h,s(infl,[],[empty]),fts(_,_))
        (you,10,m,s(n,[],[bar(2)]),fts(ag,ext))
        (did,9,h,s(comp,infl,[]),fts(_,_))
        (who,7,m,s(n,[],[bar(2),wh]),fts(_,_))
Abar chains:
 (who,0,m,s(n,[],[bar(2),wh]),fts(_,_))
A chains:
        (mary,5,m,s(n,[],[bar(2),proper]),fts(ag,ext))
        (you,2,m,s(n,[],[bar(2)]),fts(ag,ext))

.........

backtracking

      PARSE FAILED
```

REFERENCES

Abney, S. (1986). Licensing and parsing. In *Proceedings of NELS 17*, Cambridge, MA, pp. 1–15.

Abney, S. (1987). On the notion of GB parsing and psychological reality. *MIT Parsing Volume*, 1–18.

Abney, S. (1989). A computational model of human parsing. *Journal of Psycholinguistic Research 18*, 129–144.

Abney, S. and M. Johnson (1991). Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research 20*, 233–250.

Aho, A. V. and J. D. Ullman (1972). *The Theory of Parsing, Translation and Compiling.* Englewood Cliffs, NJ: Prentice-Hall.

Aho, A. V. R. S. and J. D. Ullman (1977). *Compilers: Principles, Techniques and Tools.* Reading, MA: Addison-Wesley Publishing Company.

Barton, E. (1984). Towards a principle-based parser. Technical Report 788, MIT AI Lab.

Barton, E. (1987). *The Computational Structure of Natural Language.* Ph. D. thesis, MIT, Cambridge, MA.

Barton, E., R. Berwick, and E. Ristad (1987). *Computational Complexity and Natural Language.* Cambridge, MA: MIT Press.

Belletti, A. (1988). The case of unaccusatives. *Linguistic Inquiry 19(1)*, 1–34.

Belletti, A. and L. Rizzi (1981). The syntax of *ne*: Some theoretical implications. *The Linguistic Review 1(2)*, 117–154.

Berwick, R. (1982). *Locality Principles and the Acquisition of Syntactic Knowledge.* Ph. D. thesis, MIT, Cambridge, MA.

Berwick, R. (1985). *The Acquisition of Syntactic Knowledge.* Cambridge, MA: MIT Press.

Berwick, R. (1991a). Principle-based parsing. In P. Sells, S. M. Shieber, and T. Wasow (Eds.), *Foundational Issues in Natural Language Processing*, pp. 115–226. Cambridge, MA: MIT Press.

Berwick, R. (1991b). Principle-based parsing. In R. Berwick, S. Abney, and C. Tenny (Eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, pp. 1–38. Kluwer Academic Publishers.

Berwick, R. and S. Fong (1990). Principle-based parsing. In Wilson (Ed.), *Artificial Intelligence at MIT.* Cambridge, MA: MIT Press.

Berwick, R. and A. Weinberg (1983). The role of grammars as components of language use. *Cognition 13*, 1–61.

Berwick, R. and A. Weinberg (1984). *The Grammatical Basis of Linguistic Performance.* Cambridge, MA: MIT Press.

Berwick, R. and A. Weinberg (1985). Deterministic parsing and linguistic explanation. *Language and Cognitive Processes 1(2)*, 109–134.

Bresnan, J. (1978). A realistic transformational grammar. In M. Halle, J. Bresnan, and M. George (Eds.), *Linguistic Theory and Psychological Reality*, pp. 1–59. Cambridge, MA: MIT Press.

Brody, M. (1984). On contextual definitions and the role of chains. *Linguistic Inquiry 15(3)*, 355–380.

Brody, M. (1985). On the complementary distribution of empty categories. *Linguistic Inquiry 16(4)*, 505–546.

Burzio, L. (1986). *Italian Syntax*. Dordrecht: Kluwer Academic Publishers.

Chomsky, N. (1957). *Syntactic Structures*. The Hague: Mouton.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.

Chomsky, N. (1980a). On binding. *Linguistic Inquiry 11(1)*, 1–46.

Chomsky, N. (1980b). *Rules and Representation*. Oxford: Basil Blackwell.

Chomsky, N. (1981). *Lectures on Government and Binding*. Dordrecht: Foris.

Chomsky, N. (1982). *Some Concepts and Consequences of the Theory of Government and Binding*. Cambridge, MA: MIT Press.

Chomsky, N. (1986a). *Barriers*. Cambridge, MA: MIT Press.

Chomsky, N. (1986b). *Knowledge of Language: Its Nature, Origin and Use*. New York: Praeger.

Chomsky, N. (1988). Some notes on economy of derivation and representation. *MIT Working Papers in Linguistics 10*, 43–74.

Chomsky, N. (1992). A minimalist program for linguistic theory. *Occasional MIT Working Papers in Linguistics 1*. (also appeared in *The View from Building 20*, edited by Ken Hale and Jay S. Keyser. MIT Press, Cambridge, MA, 1993, 1-52.).

Chomsky, N. and M. Halle (1968). *The Sound Pattern of English*. New York: Harper and Row.

Church, K. (1980). On memory limitations in natural language processing. Technical Report TR 245. MIT/LCS.

Cinque, G. (1990). *Types of $\bar{A}$ Dependencies*. Cambridge, MA: MIT Press.

Clark, R. (1990). Chain formation. University of Geneva.

Clifton, C., L. Frazier, and C. Connine (1984). The use of syntactic information in filling gaps. *Journal of Verbal Learning and Verbal Behaviour 23*, 696–708.

Cole, R. and J. Jakimik (1978). Understanding speech. In G. Underwood (Ed.), *Strategies of Information Processing*. London: Academic Press.

Comrie, B. (1981). *Language Universals and Linguistic Typology*. Oxford: Basil Blackwell.

Connine, C. M., F. Ferreira, C. Jones, C. Clifton, and L. Frazier (1984). Verb frame preferences: Descriptive norms. *Journal of Psycholinguistics Research 13*, 307–319.

Correa, N. (1988). *Syntactic Analysis of English with Respect to Government-Binding Theory*. Ph. D. thesis, Syracuse University, Syracuse, NY.

Correa, N. (1991). Empty categories, chain binding and parsing. In R. Berwick, S. Abney, and C. Tenny (Eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, pp. 83–122. Dordrecht: Kluwer Academic Publishers.

Crain, S. and J. D. Fodor (1985). How can grammars help parsers? In D. Dowty, L. Kartunnen, and A. Zwicky (Eds.), *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, pp. 94–128. Cambridge: Cambridge University Press.

Crocker, M. (1992). *A Logical Model of Competence and Performance in the Human Sentence Processor.* Ph. D. thesis, University of Edinburgh, Edinburgh, Scotland.

Crocker, M. (1995). *Computational Psycholinguistics: A Interdisciplinary Perspective.* Dordrecht: Kluwer Academic Publishers.

De Marcken, C. (1990). Parsing the LOB corpus. In *Proceedings of the 28th Meeting of the Association for Computational Linguistics*, Pittsburgh, PA, pp. 243–251.

De Vincenzi, M. (1991). *Syntactic Parsing Strategies in Italian.* Dordrecht: Kluwer Academic Publishers.

Dorr, B. J. (1987). UNITRAN: a principle-based approach to machine translation. Technical Report 100, AI Lab, MIT, Cambridge, MA.

Dorr, B. J. (1990). *Lexical Conceptual Structure and Machine Translation.* Ph. D. thesis, MIT, Cambridge, MA.

Dorr, B. J. (1993). *Machine Translation: A View from the Lexicon.* Cambridge, MA: MIT Press.

Downing, B. T. (1978). Some universals of relative clause structure. In G. J. H. (Ed.), *Universals of Human Language*, pp. 375–418. Stanford, CA: Stanford University press.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery 14*, 453–460.

Ferreira, F. and C. Clifton (1986). The independence of syntactic processing. *Journal of Memory and Language 25*, 348–368.

Ferreira, F. and J. M. Henderson (1990). The use of verb information in syntactic parsing: A comparison of evidence from eye-movements and word-by-word self-paced reading. *Journal of Experimental Psychology: Learning, Memory and Cognition 16*, 555–568.

Fodor, J. A., T. Bever, and M. Garrett (1974). *The Psychology of Language: An Introduction to Psycholinguistics and Generative Grammar.* New York: McGraw-Hill.

Fodor, J. D. (1985). Deterministic parsing and subjacency. *Language and Cognitive Processes 1(1)*, 3–42.

Fodor, J. D. (1989). Empty categories in sentence processing. *Language and Cognitive Processes 3-4, SI*, 155–209.

Fong, S. (1990). Free indexation: Combinatorial analysis and a compositional algorithm. *Proceedings of the 28th Meeting of the Association for Computational Linguistics*, 105–110.

Fong, S. (1991). *Computational Properties of Principle-based Grammatical Theories.* Ph. D. thesis, MIT, Cambridge, MA.

Fong, S. and R. Berwick (1992). Isolating cross-linguistic parsing complexity with a principle-and parameters parser: a case study of Japanese and English. In *Proceedings of COLING 92*, Nantes, pp. 631–637.

Ford, M., J. Bresnan, and R. Kaplan (1982). A competence-based theory of syntactic closure. In J. Bresnan (Ed.), *The Mental Representations of Grammatical Relations*, pp. 727–796. Cambridge, MA: MIT Press.

Forster, K. and I. Olbrei (1973). Semantic heuristics and syntactic analysis. *Cognition 2(3)*, 319–347.

Frampton, J. (1990). Parasitic gaps and the theory of wh-chains. *Linguistic Inquiry 21(1)*, 49–77.

Frank, R. (1990). Licensing and tree adjoining grammar in GB parsing. In *Proceedings of the 28th Meeting of the Association for Computational Linguistics*, Pittsburgh, PA, pp. 111–117.

Frank, R. (1991). Two types of locality in government and binding parsing. manuscript, University of Pennsylvania, Philadeplhia, PA.

Frank, R. (1992). *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives.* Ph. D. thesis, University of Pennsylvania, Philadelphia, PA.

Frazier, L. (1989). Against lexical generation of syntax. In W. Marslen-Wilson (Ed.), *Lexical Representation and Processes*, pp. 505–528. Cambridge, MA: MIT Press.

Frazier, L. and C. Clifton (1989). Successive cyclicity in the grammar and the parser. *Language and Cognitive Processes 4*, 93–126.

Frazier, L., C. Clifton, and J. Randall (1983). Filling gaps: Decision principle and structure in sentence comprehension. *Cognition 13*, 187–222.

Frazier, L. and K. Rayner (1982). Making and correcting errors during sentence comprehension: Eye-movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology 14*, 178–210.

Frazier, L. and K. Rayner (1987). Resolution of syntactic category ambiguities: Eye movements in parsing lexically ambiguous sentences. *Journal of Memory and Language 26*, 505–526.

Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985). *Generalized Phrase Structure Grammar.* Oxford: Blackwell.

Gibson, E. (1991). *A Computational Theory of Human Linguistic Processing: Memory Limitations and Processing Breakdown.* Ph. D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Gorrell, P. (1991). Subcategorization and sentence processing. In R. Berwick, S. Abney, and C. Tenny (Eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, pp. 279–300. Cambridge, MA: MIT Press.

Grimshaw, J. (1986). Subjacency and the S/S$'$ parameter. *Linguistic Inquiry 17(2)*, 364–369.

Haegeman, L. (1991). *Introduction to Government and Binding.* Oxford: Basil Blackwell.

Hawkins, J. (1990). Word order universals. *Linguistic Inquiry 21(2)*, 223–261.

Holmes, V. M., A. Kennedy, and W. Murray (1987). Syntactic structure and the garden path. *The Quarterly Journal of Experimental Psychology 39A*, 277–293.

Holmes, V. M., L. Stowe, and L. Cupples (1989). Lexical expectations in parsing complement-verb sentences. *Journal of Memory and Language 28*, 668–689.

Irons, E. (1961). A syntax directed compiler for ALGOL 60. *Commmunications of the Association for Computing Machinery 4*, 51–55.

Jackendoff, R. (1983). *Semantics and Cognition.* Cambridge, MA: MIT Press.

Jackendoff, R. (1990). *Semantic Structures.* Cambridge, MA: MIT Press.

Jaeggli, O. and K. Safir (1989). *The Null Subject Parameter.* Kluwer Academic Publishers.

Johnson, M. (1989). The use of knowledge of language. *Journal of Psycholinguistic Research 18.1*, 105–129.

Joshi, A. (1985). How much context-sensitivity is required to provide reasonable structural descriptions: Tree adjoining grammars. In Z. Dowty and Kartunnen (Eds.), *Natural Language Processing: Psycholinguistic, Computational and Theoretical Perspectives*, pp. 206–250. Cambridge, MA: Cambridge University Press.

Kashket, M. (1991). *A Parameterized Parser for English and Warlpiri*. Ph. D. thesis, MIT, Cambridge, MA.

Kayne, R. (1994). *The Antisymmetry of Syntax*. Cambridge, MA: MIT Press.

Kimball, J. (1973). Seven principles of surface structure parsing in natural language. *Cognition 2(1)*, 15–47.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control 8*, 607–639.

Knuth, D. E. (1968). Semantics of context-free languages. *Mathematical Systems Theory 2*, 127–145.

Knuth, D. E. (1973). *The Art of Computer Programming*, Volume 3. Menlo Park, CA: Addison-Wesley Publishers.

Kornai, A. (1983). $\overline{\text{X}}$ grammars. In S. J. Bolyai (Ed.), *Algebra, Combinatorics and Logic in Computer Science*, pp. 523–536. Gyor, Hungary: Colloquia Mathematica.

Kuno, S. (1974). The position of relative clauses and conjunction. *Linguistic Inquiry 5*, 117–136.

Laenzlinger, C. (1993). Principles for a formal account of adverb syntax. *Geneva Generative Papers 1(2)*, 47–75.

Lang, B. (1974). Deterministic techniques for efficient nondeterministic parsers. In *Proceedings of the Second Colloquium on Automata, Languages and Programming*, pp. 255–269.

Lang, B. (1989). Towards a uniform formal framework for parsing. In M. Tomita (Ed.), *Current Issues in Parsing Technologies*, pp. 153–172. Dordrecht: Kluwer Academic Publishers.

MacDonald, M. (1994). Probabilistic constraints and syntactic ambiguity resolution. *Language and Cognitive Processes 9(2)*, 157–201.

Macias, B. (1991). *An Incremental Parser for Government Binding Theory*. Ph. D. thesis, Cambridge University, Cambridge, MA.

Marcus, M. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge, MA: MIT Press.

Marcus, M., D. Hindle, and M. Fleck (1983). D-theory: Talking about talking about trees. In *Proceedings of the 21st Meeting of the Association for Computational Linguistics*, Morristown, NJ, pp. 129–136.

Marr, D. (1982). *Vision*. S.Francisco, CA: W.H.Freeman.

Marslen-Wilson, W. (1973). Linguistic structure and speech shadowing at very short latencies. *Nature 244*, 522–523.

Merlo, P. (1994). A corpus-based analysis of verb continuation frequencies. *Journal of Psycholinguistics Research 23(6)*, 435–457.

Miller, G. and N. Chomsky (1963). Finitary models of language users. In R. R.Luce and E.Galanter (Eds.), *Handbook of Mathematical Psychology*, Volume 2, pp. 419–491. New York.

Miller, G. A. and S. Isard (1964). Free recall of self-embedded english sentences. *Information and Control* 7, 292–303.

Mitchell, D. (1987). Lexical guidance in human parsing: Locus and processing characteristics. In M. Coltheart (Ed.), *Attention and Performance XII: The Psychology of Reading*, pp. 601–618. Hillsdale, NJ: Erlbaum.

Mitchell, D. and V. Holmes (1985). The role of specific information about the verb in parsing sentences with local structural ambiguity. *Journal of Memory and Language 24*, 542–559.

Nozohoor-Farshi, R. (1986). On formalizations of the marcus's parser. In *Proceedings of COLING 86*, Bonn, pp. 533–535.

Pereira, F. and D. Warren (1981). Parsing as deduction. In *Proceedings of the 19th Meeting of the Association for Computational Linguistics.*

Pereira, F. and R. Wright (1991). Finite state approximation of phrase structure grammars. In *Proceedings of the 29th Meeting of the ACL*, Berkeley, CA, pp. 246–255.

Pesetsky, D. (1987). *Wh* in situ: Movement and unselective binding. In E. Reuland and A. ter Meulen (Eds.), *The Representation of (In)definiteness.* Cambridge, MA: MIT Press.

Phillips, J. (1992). A computational representation for Generalized Phrase Structure Grammars. *Linguistics and Philosophy 15(3)*, 255–287.

Phillips, J. and H. Thompson (1986). GPSGP: A parser for Generalized Phrase Structure Grammars. *Linguistics 23.*

Pollock, J.-Y. (1989). Verb movement, universal grammar and the structure of IP. *Linguistic Inquiry 20(3)*, 365–424.

Pritchett, B. (1992). *Linguistic Competence and Parsing Performance.* Chicago, IL: University of Chicago Press.

Rayner, K. and L. Frazier (1987). Parsing temporarily ambiguous complements. *The Quarterly Journal of Experimental Psychology 39A.* 657–673.

Reinhart, T. (1976). *The Syntactic Domain of Anaphora.* Ph. D. thesis, MIT, Cambridge, MA.

Ristad, E. (1990). Computational structure of human language. Technical Report 1260, MIT AI Lab, Cambridge, MA.

Rizzi, L. (1982). *Issues in Italian Syntax.* Dordrecht: Foris.

Rizzi, L. (1990). *Relativized Minimality.* Cambridge, MA: MIT Press.

Rizzi, L. (1991). Residual verb second and the wh-criterion. Technical Report 2, University of Geneva. Technical Reports in Formal and Computational Linguistics.

Ross, J. (1967). *Constraints on Variables in Syntax.* Ph. D. thesis, MIT.

Rounds, W. (1991). The relevance of computational complexity theory to natural language processing. In P. Sells, S. M. Shieber, and T. Wasow (Eds.), *Foundational Issues in Natural Language Processing*, pp. 9–30. Cambridge, MA: MIT Press.

Schabes, Y. (1991). Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In *Proceedings of the 29th Meeting of the ACL*, Berkeley, CA, pp. 106–113.

Schabes, Y. and K. Vijay-Shanker (1990). Deterministic left to right parsing of tree adjoining languages. In *Proceedings of the 28th Meeting of the ACL*, Pittsburgh, PA., pp. 276–283.

Seidenberg, M., M. Tanenhaus, J. Leiman, and M. Bienkowski (1982). Automatic access of the meaning of ambiguous words in context: Some limitations of knowledge- based processing. *Cognitive Psychology 14*, 489–537.

Shieber, S. M. (1984). Direct parsing with ID/LP grammars. *Linguistics and Philosophy 7(2)*, 135–154.

Shieber, S. M. (1986). A simple reconstruction of GPSG. In *Proceedings of COLING 86*, pp. 211–215.

Shieber, S. M. and M. Johnson (1993). Variations on incremental interpretations. *Journal of Psycholinguistic Research 22(2)*, 287–319.

Shopen, E. T. (1985). *Language Typology and Syntactic Description.* Cambridge, MA: Cambridge University Press.

Slobin, D. I. (1966). Grammatical transformations and sentence comprehension in childhood and adulthood. *Journal of Verbal Learning and Verbal Behaviour 5*, 219–227.

Stabler, E. (1990). Relaxation techniques for principle-based parsing. Talk given at the Workshop on GB Parsing, University of Geneva.

Stabler, E. (1991). Avoid the pedestrian's paradox. In R. Berwick, S. Abney, and C. Tenny (Eds.), *Principle-based Parsing: Computation and Psycholinguistics*, pp. 199–238. Dordrecht: Kluwer Academic Publishers.

Stabler, E. (1992). *The Logical Approach to Syntax.* Cambridge, MA: MIT Press.

Stabler, E. (1994). Left corner parsing for incremental interpretation. manuscript, UCLA.

Steedman, M. (1989). Grammar, interpretation and processing from the lexicon. In W. Marslen-Wilson (Ed.), *Lexical Representation and Processes*, pp. 463–504. Cambridge, MA: MIT Press.

Steele, S. (1978). Word order variation. In G. J. H. (Ed.), *Universals of Human Language*, pp. 585–623. Stanford, CA: Stanford University Press.

Stevenson, S. (1994). *A Competitive Attachment Model for resolving Syntactic Ambiguities in Natural Language Parsing.* Ph. D. thesis, University of Maryland at College Park. also available as RuCCs TR 18, Rutgers Center for Cognitive Science.

Stowe, L. (1986). Evidence for on-line gap location. *Language and Cognitive Processes 1*, 227–245.

Stowell, T. (1981). *Origins of Phrase Structure.* Ph. D. thesis, MIT, Cambridge, MA.

Sturt, P. and M. Crocker (1995). Incremental parsing. CUNY Conference, Tucson, Arizona.

Swinney, D. (1979). Lexical access during sentence comprehension. *Journal of Verbal Learning and Verbal Behaviour 18*, 645–660.

Thompson, H. (1982). Handling metarules in a parser for GPSG. Technical Report 175, Department of Artificial Intelligence, University of Edinburgh.

Tomita, M. (1985). *Efficient Parsing for Natural Language.* Hingham, MA: Kluwer.

Tomita, M. (1986). *Efficient Parsing of Natural Language.* Dordrecht: Kluwer Academic Publishers.

Tomita, M. (1987). An efficient augmented context-free parsing algorithm. *Computational Linguistics 13(1-2)*, 31–46.

Trueswell, J., M. Tanenhaus, and C. Kello (1993). Verb specific constraints in sentence processing: Separating effects of lexical preference from garden-paths. *Journal of Experimental Psychology: Learning, Memory and Cognition 19*, 528–553.

Van de Koot, H. (1990). *Essay on the Grammar-Parser Relation.* Dordrecht: Foris.

Van de Koot, H. (1991). Parsing with principles: on constraining derivations. *UCL Working Papers in Linguistics*, 369–396.

Wanner, E. and M. Maratsos (1978). An atn approach to comprehension. In H. Morris, J. Bresnan, and G. Miller (Eds.), *Linguistic Theory and Psychological Reality*, pp. 119-161. Cambridge, MA: MIT Press.

Wehrli, E. (1992). The IPS system. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, pp. 870-874.

Weinberg, A. (1988). *Locality Principles in Syntax and in Parsing*. Ph. D. thesis, MIT, Cambridge, MA.

Weinberg, A. (1993). Parameters of the theory of sentence processing: Minimal committment theory goes east. *Journal of Psycholinguistics Research 22(3)*, 339-363.

Woods, W. (1970). Transition network grammars for natural language analysis. *Communications of the Association for Computing Machinery 13*, 591-606.

INDEX