Computational Properties of Principle-Based Grammatical Theories

by

Sandiway Fong

B.Sc.(Eng), Computing Science, Imperial College of Science and Technology
University of London
(1984)

S.M. Electrical Engineering and Computer Science, Massachusetts Institute of Technology
(1986)

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

at the
Massachusetts Institute Of Technology
June 1991

Signature of Author______________________________

Department of Electrical Engineering and Computer Science
May 20th 1991

Certified by______________________________

Robert C. Berwick
Associate Professor of Computer Science and Engineering,
Artificial Intelligence Laboratory,
Thesis Supervisor

Accepted by______________________________

Arthur C. Smith
Chairman, Departmental Graduate Committee

Computational Properties of Principle-Based Grammatical Theories

by

Sandiway Fong

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering and Computer Science.

Abstract

Recently there has been a conceptual shift to modular theories of syntax in linguistics. Instead of being based on complex and large rule systems, these theories rely on the interaction of a small set of universal principles that are parameterized across languages. These theories pose many new and difficult problems for the design of parsing systems. The major issue tackled in this thesis is how to achieve substantial linguistic coverage and efficient parsing whilst maintaining a level of representation of principles close to that used in the linguistics literature.

This thesis describes a principle-based parsing system that achieves substantial coverage. Because of the free and complex interaction between principles, one major barrier to the use of these theories has been the problem of finding a comprehensive, yet consistent set of principles. Using a small set of twenty-five principles, the system has been demonstrated to correctly account for hundreds of different constructions from an introductory linguistics textbook. Also, illustrating the universal nature of principle-based theories, the same set of principles has been demonstrated to cover examples of Japanese as well as English data.

This thesis also investigates the problem of configuring principles for efficient parsing. The parsing system incorporates flexible parameters of control independent from principle definitions. These control parameters effectively define a family of parsers that incorporate the same linguistic knowledge, but with different performance characteristics. By investigating the effect of variations in control settings, we obtain a characterization of the relevant computational properties of principles that determine the most appropriate control configurations for efficient parsing.

Lastly, this thesis investigates the problem of representing principles at a level that is readily accessible to the linguist. Given the continual revision of linguistic theories, parsing systems must be updated frequently to track the revisions and advances in linguistic theory. By adopting a high-level, logic-based representation, together with the use of automatic compilation techniques, the system tackles the problem of bridging the gap between an abstract and an efficiently executable representation. This approach simplifies the necessary task of formalization and provides an answer to the parser obsolescence problem.

Thesis Supervisor: Prof. Robert C. Berwick
Title: Professor of Computer Science and Engineering

To See Seow Chu

Acknowledgments

This thesis is the product of countless, but occasionally productive hours spent staring at a computer console, fortunately relieved by invaluable and stimulating discussions with many people at MIT. In particular, I would like to extend my gratitude to the following individuals:

> My advisor, Bob Berwick, who provided constant encouragement and prodded for answers to many hard questions.
>
> The linguists with whom I had the opportunity to discuss various aspects of this project and those who patiently rectified many of my misconceptions and inadequate knowledge of syntactic theory: Howard Lasnik, Alec Marantz, David Pesetsky, Mamoru Saito, Noam Chomsky, Carol Tenny, Miori Kubo and Doug Jones.
>
> Certain members of the AI Lab. for many hours of both intellectual and non-intellectual conversations at odd hours of the morning, chiefly: Eric Ristad and Peng Wu.

Some sections from various chapters have appeared separately as papers. Section 3.4.2 is taken from "Free Indexation: Combinatorial Analysis and a Compositional Algorithm" in *ACL-90.* Chapter 5 is a much expanded version of "The Computational Implementation of Principle-Based Parsers" in *1st International Workshop on Parsing Technologies* 1989, in the book *Current Issues in Parsing Technologies*, ed. M. Tomita, Kluwer 1990, and partly in chapter 12 of *Artificial Intelligence at MIT: Expanding Frontiers. Volume 1.* ed. P.H. Winston. MIT Press. 1990. Sections 6.1 and 6.2 are taken from "Principle-Based Parsing and Type Inference." in *3rd International Workshop on Natural Language Understanding and Logic Programming.* 1991, and will also appear as a chapter in the book *Natural Language and Logic Programming III*, eds. Brown & Koch. North Holland. 1991.

Contents

CHAPTER 1

Introduction

This thesis describes a natural language parser that:

- Is based on modern principle-based grammatical theories rather than traditional rule-based systems. Principle-based theories can replace thousands of traditional rules using a small number of fundamental linguistic principles. The implemented system employs only twenty-five principles plus a small number of (about thirty) phrase structure rules.

- Implements a more comprehensive linguistic theory than existing principle-based systems. The system achieves its goal of systematically covering hundreds of different examples from an introductory linguistics textbook, *A Course in GB Syntax* by Lasnik & Uriagereka [35].

- Has the ability to handle different languages using the same core set of universal principles despite seemingly large typological differences. For instance, the parser can be parameterized to handle either English or Japanese by specifying an appropriate parameter vector and lexicon.

- Permits the construction of a family of parsers which make use of the same set of principles (and therefore make the same grammaticality judgements), but with varying parsing times for different sentences. The system allows the user to alter simple parameters of control, independent from principle definitions, that can greatly affect the time spent parsing. For instance, the order in which principles are applied is subject to user control. Another option, called *principle interleaving*, allows a principle to be co-routined so as to apply to partial phrase structures as they are built, or alternatively, to wait until structure for a complete sentence has been recovered.

- Uses a high-level, logic-based representation allowing principle definitions that closely resemble the natural language definitions found in the linguistics literature to be constructed.

- Employs a variety of compilation techniques including canonical LR(1)-based parser-generation, type inference, simple partial evaluation, and program transformation to tackle the problem of automatically and transparently bridging the gap between an abstract and an efficiently executable representation. Of the above techniques, we will briefly expand on the two most interesting:

    1. The use of a canonical LR(1)-based algorithm for recovering phrase structures is particularly noteworthy. Although the algorithm is generally superior to comparable methods; in particular, with respect to the use of lookahead, very large tables have to be constructed for non-trivial grammars. This has precluded its use in natural language parsing as well as for programming languages. However, the relatively small size of the phrase structure grammar used here makes canonical LR(1)-based parsing eminently practical, as will be discussed in chapter 4.
    2. Another interesting technique is the use of type inference methods, based on a novel definition of the "type" of a linguistic principle, to safely reduce the amount of work required to apply principles as early as possible in the principle interleaving framework mentioned earlier.

- Provides a possible answer to the *parser obsolescence problem*; that is, the problem of updating parsers to keep track of frequent revisions and advances in linguistic theory. Because of the small grammar size, the high-level nature of the representation, the separation between principle definitions and control options, and the presence of automatic compilation procedures, the necessary tasks of formalizing definitions, ensuring correctness, and generating efficient parsers can be made much less tedious.

1.1 Outline of the Introduction

The following sections of this chapter is organized around the following topics:

- *Linguistic principles.*

  To introduce the notion of a linguistic principle and other concepts employed by principle-based syntactic theories, we will begin by examining what we can expect such theories to tell us about the interpretation of sentences.

- *Problems for principle-based systems.*

  We will then briefly list some of the major obstacles that must be overcome if principle-based parsers are to become widely adopted.

The major question addressed in this thesis is one of general parser design; that is, how to achieve substantial linguistic coverage and maintain efficient parsing, whilst retaining a perspicuous representation of grammar that can be readily maintained or updated? The remaining sections in this chapter will summarize how the implemented system tackles this question.

- *Linguistic coverage.*

  First, we will review the linguistic coverage of the implemented system. As noted earlier, the system has been extensively tested on a wide range of linguistic data found in Lasnik & Uriagereka. Using snapshots of the system in action, we will cover examples of well-formed input, both ambiguous and non-ambiguous. We will also illustrate how the parser explains ill-formed input in terms of the underlying principles. Furthermore, to provide an idea of the flexibility of the implemented theory, we will describe how the same system can also handle Japanese sentences. We will introduce the notion of language parameters by describing some of the important typological distinctions between English and Japanese. These distinctions will be highlighted using actual parses produced by the system on Japanese examples taken from the linguistics paper *On the Nature of Proper Government* by Lasnik & Saito [34].

- *Control options and parsing efficiency.*

  Secondly, we will describe the control options that the current system makes available to the user; namely, the freedom to shuffle the order in which principles are processed and the option of incrementally applying selected principles to partial structures as phrase structure construction proceeds. All possible control choices have the important property that they cannot affect the linguistic judgements rendered by the parser — only the amount of work that has to done to reach a particular judgement. By running the same sentence on two particular settings, we will demonstrate the large effect that these control parameters can have on parsing time. From this one example, we will expand the discussion to address the general problem of configuring the parser for efficient processing. Here, we will summarize the relevant results of chapters 5 and 6 by describing the computationally relevant features of individual principles that help to determine the appropriate control settings. Because control choices are transparently and automatically reflected in the parser (with respect to principle definitions), parser control can also be easily updated (as necessary) as principle definitions are revised. We will also take the opportunity to list some practical implementation issues that will turn out to be important factors in determining the effectiveness of the various options.

- *Implementation.*

> Next, we will provide simple examples of how principles may be perspicuously represented in a notation that closely resembles the definitions found in the linguistics literature. An important feature of the representation is that principle definitions can be transparently targeted for different parser configurations, i.e. the grammar writer is free to specify principles that are readily comprehensible without needing to know exactly how principles are realized or combined. A large portion of the system is devoted to maintaining this separation. Type inference and program transformation are among the methods employed to automatically compile principles into the underlying forms actually used to parse. We will briefly review the major components and general organization of the grammar compilation system.

Finally, we will end the introduction with an outline of the main chapters of the thesis.

1.2 Linguistic Principles

What do we mean when we say that a computer program can "parse" sentences? For example, what can a syntax tree tell us about a particular sentence? To answer these questions, let us consider what is involved when we say we understand a sentence such as (1):

(1) Which report did you file without reading?

To interpret this, we must know some simple facts about the words in the sentence: for example, that *file* is a predicate that takes two arguments, an 'agent' that does the filing and an element to be filed; similarly for *reading*, that there is an agent to do the reading and an element that is being read. Apart from the import of individual words, we must also understand that the element to be filed and read (or not read, for that matter) are one and the same object. Furthermore, this object must be construed, or be *coreferent*, with the 'report' that was mentioned at the beginning of the sentence. In a parallel fashion, it is also understood that the agent doing the filing, the (potential) reader, and the person denoted by the pronoun *you* must be one and the same person. Finally, we understand that no other interpretation is possible; that is, the sentence is unambiguous.

Modern syntactic theories can identify such coreference relations and the semantic, or *thematic*, roles that syntactic elements such as noun phrases and clauses play in a sentence. In addition to identifying the necessary elements for interpretation, such theories are capable of not only distinguishing between similar-looking sentences with radically different grammatical status, but also to provide a principled explanation for that difference. For example, such theories can also account for the following three counterparts to (1):

(2) a. *The report was filed without reading?[1]
b. *Who filed which report without reading?
c. Which report did you file without reading it?

A surprising property of some of these theories is that they can account for a wide variety of facts using a relatively small set of fundamental assumptions, or *principles*. What do these principles look like? And what are the kinds of objects that such principles operate on? Although it is beyond the scope of this introduction to fully explain how the theory described in Lasnik & Uriagereka accounts for the above data (see chapter 3 in Lasnik & Uriagereka for the details), we will briefly introduce some of the relevant principles that are crucial to the analysis. (The following description will also serve as an introduction to linguistic concepts that will be relevant to the parsing examples presented later in the chapter.)

- *Empty categories.*

  In general, thematic arguments such as the ones we have mentioned above must be represented in syntactic structure, either by overt or non-overt elements. For example, the position occupied by the argument *it* in (2c) is also assumed to exist in the parallel example (1). This so-called "gap" will be occupied by an *empty category* (EC). An important point to note is that, in many theories, these ECs will be treated as "first-class syntactic elements" in a sense to be made clear immediately below.

- *Binding conditions.*

  Binding conditions are principles that determine the coreference possibilities for elements such as anaphors, e.g. *himself*, or pronouns, e.g. *him* or *it* in (2c):

  (A) *An anaphor must be coreferent with an element in its clause.*
  (B) *A pronoun must* ***not*** *be coreferent with any element in its clause.*

  Even in this simplest form, such conditions are powerful enough to explain a variety of facts:

  (3) a. *[John$_1$ likes him$_1$] — *cf.* [John$_1$ likes him$_2$]
  b. John$_1$ believes that [Mary likes him$_1$]
  c. John$_1$ believes that [Mary likes him$_2$]
  d. [John$_1$ likes himself$_1$] — *cf.* *[John$_1$ likes himself$_2$]
  e. John believes that [Mary$_1$ likes herself$_1$]
  f. *John$_1$ believes that [Mary likes himself$_1$]

  (Here we have used brackets to delimit the extent of the relevant clause and simple indices to distinguish the various interpretations.)

[1]Note: it is standard notation is to use an asterisk '*' to mark ungrammatical sentences.

Generally, noun phrases will be classified in terms of the independent binary features [±A] (anaphoric) and [±P] (pronominal). For example, *himself* and *him* are assumed to have the features [+A,−P] and [−*A*,+P], respectively. Referential expressions such as names, e.g. *John*, will be classified as [−A,−P]. These features are extended in a uniform manner to ECs that occupy the same subject and object positions as the overt pronouns, anaphors and names shown in (3). Binding conditions (A) and (B) will apply to overt and non-overt elements bearing the feature values [+A] and [+P], respectively. However, one question immediately arises: what are the appropriate values for ECs? The general rule that applies in such cases will be the null hypothesis; that is, all options should be freely available. The essential idea is that all but the appropriate option will be ruled out by the interaction of principles. For example, the EC that represents the object of *reading* in (1) cannot have value [+A] because it would violate Binding condition (A). The concept of free assignment operating under interacting constraints is a general and pervasive characteristic of many theories. Note that the interaction of constraints need not result in unique assignments, e.g. as in the case of coreference relations. For example, a pronoun can be ambiguous in the sense that it may have more than one possible referent, as illustrated by (3b) and (3c). The relevant principle in this case, condition (B), admits both possibilities.[2]

- *Functional determination.*

  Functional determination is another important constraint on the free assignment of [±A,±P] features to ECs. For the purpose of this discussion, we will be concerned with ECs that have feature values [−A,−P]. Such ECs are termed *variables* because of certain similarities to variables in standard logic that will be made apparent below. If we assume that the result of syntactic analysis will be some kind of *Logical Form* (LF) that will serve as the input to interpretation, then variables are elements that are bound by operators, e.g. a *wh*-word like *what* or *why*, or quantifiers, e.g. *someone*. For example, a LF representation for (1) will be of the form:

  (4) for which x, x a report, you file x without reading x

  In general, an EC that is most closely associated with an operator will be *functionally determined* as a variable, i.e. have feature values [−A,−P]. In particular, to be interpreted as a variable, an EC must occur within the scope

[2] Note that condition (B) also predicts that *it* in (2c) is ambiguous. In this case, the more obvious interpretation is for the pronoun to refer to the 'report'. However, a second interpretation is also available. Consider the following scenario: *"Did you get my 'Ten Rules of Filing'? Which report did you file without reading it?"*

> of an operator, in much the same way that a logical variable x must occur within the scope of a $\forall x$ or $\exists x$ in standard logic. In the case of (1), the objects of *file* and *reading* will be uniquely determined as variables so that (4) will be the only possible LF. This licensing restriction also explains why the empty object of *reading* in the similar example (2a) cannot be interpreted as a variable. Assuming that: (1) all other assignments of [$\pm$A,$\pm$P] are ruled out in both cases — we will not go into the details here, and (2) such an EC *must* bear some assignment of the [$\pm$A,$\pm$P] features; then functional determination has accounted for the contrast in grammaticality between the two examples.

To summarize, we have briefly seen how a modern syntactic theory can explain a variety of phenomena using just a few assumptions motivated by semantic interpretation concerns. Consider for a moment how one might account for the same facts in traditional rule-based systems such as context-free grammars (CFGs) or early transformational grammars (TGs), or computational formalisms such as the augmented transition networks (ATNs)? Barton [4] has described in detail the many scientific and engineering problems of such frameworks. We will not replicate those arguments here, save to mention that most formalisms seemed to suffer from a combination of the following problems: (1) it seemed descriptively necessary to have a large number of rules; (2) individual rules seemed to be overly detailed and complex, in some cases the unconstrained framework makes system extension difficult; and (3) because of the "closeness" to surface forms, rules were highly language-specific. Given these disadvantages, a descriptively adequate formulation in such frameworks (assuming it would be possible to do so) would certainly not be as succinct, nor have the explanatory force of the principle-based approach. Note also that the very general nature of the constraints we have described suggests that such theories are not restricted to any one particular language. Note also that there is no mention of word order, for instance, in any of them. In fact, linguists have shown that such principles, albeit with minor parametric variations, do indeed apply to a variety of different languages (as we will illustrate in section 1.4). Given all these advantages, we might naturally ask the question:

> Why, then, have there been so few natural language systems built using principle-based theories of grammar?

In the next section, we will review possible reasons why this might be the case. Then, from these and other considerations, we will provide an overview of how specific problems posed by the linguistics framework are tackled in a parser that uses principles to parse sentences such as the ones we have considered above.

1.3 Problems with Principles

Let us now turn to list some possible objections to principle-based system in general:

- *Principles are too vague, or underspecified. They cannot be formalized.*

  Most principles are stated incompletely, and sometimes ambiguously, using natural language (as opposed to some rigid mathematical formalism). Since a precise formulation of knowledge is normally considered to be a necessary first step towards use of that knowledge; it is not surprising that proper formalization is considered to be a difficult problem for potential grammar writers. However, Stabler [52] has painstakingly described how the *Barriers* framework of Chomsky [11] (by no means a complete theory, but nonetheless a good indication of the kind of theory that would be required for a "full" parser), may be axiomatized in first order logic. Of course, other issues such as that of correctness and completeness naturally arise from any formalization. We will return to consider these questions in chapter 2.

- *Principle-based theories are not logically consistent. Hence, even if they can be formalized, they cannot be used to parse.*

  Principles are naturally organized into *modules*, each dealing with a different aspect of syntax. For example, the *Binding conditions* mentioned earlier are part of *Binding theory* which deals with general conference conditions. The identification of thematic roles, e.g. 'agent', with positions in syntactic structure is part of *Theta (θ) -theory* which governs the proper distribution of thematic roles. Many theories actually focus on only a few modules; hence leaving open the possibility for conflicts. Moreover, linguists often propose several different accounts, sometimes within the same paper, without settling on any one particular version. Of course, to avoid an overly under-constrained system, the grammar writer will be forced to assemble a comprehensive set of modules.

- *There are too many gaps in coverage for current theories to be practical.*

  It is a fact that all existing linguistic frameworks are descriptively incomplete. However, coverage in current principle-based theories is somewhat uneven. Although such theories are capable of explaining linguistically sophisticated examples that are relatively uncommon in everyday usage, e.g. the sentences shown in (1) and (2), they cannot handle unrestricted text.[3]

[3] Note that systems with goals to "parse" arbitrary text do exist (see Fujisaki [25] and de Marcken [19] for examples of recent work). Such systems are typically based on traditional rules, sometimes augmented with statistical methods. However, it is not clear that such systems will be able to handle sentences such as the ones that we have been discussing, nor recover as much in the way of linguistic information.

- *Current implementations are "toy" systems.*

  Although nothing near what could be called a complete set of principles is currently available, most initial efforts can be characterized as making use only of a very restricted subset of the available principles. The lack of a readily-available formal specification of a fairly large and consistent set of principles is probably the major contributing factor here.

- *Principle-based systems are too slow.*[4]

  Conventional systems based on, say, context-free (CF) rules have been able to draw upon the many efficient methods that were initially developed for fast parsing of computer programs. By comparison, the principle-based systems that have been built are certainly much slower. Barton [4] notes that Fodor has argued that modular systems are at a disadvantage because they must access and integrate information from more than one source.[5] We note here that there has been relatively little in the way of direct comparisons between different ways of organizing the same principles for efficient processing. Moreover, current systems have not always exploited specific computational properties of individual principles to obtain efficient interpretation or compilation methods.

The above list is by no means an exhaustive catalogue of the potential problems, but it should give the reader some indication of the (very real) difficulties that challenge parser designers. We will return to expand on these (and many other relevant issues for parser design) in chapter 2, where we will take the opportunity to contrast the approach taken in this thesis with other existing systems. However, in the next section, we will focus on the current implementation. We believe that the system goes a long way towards demonstrating that (relatively) large scaled and sophisticated principle-based parsing systems can be built. As we shall see, principles from many modules can be formalized and shown to be consistent insofar as many hundreds of examples taken from Lasnik & Uriagereka have been "correctly", or faithfully, accounted for. The operational notion of *correctness* that we will adopt in this thesis is that the parser must recover the same structure (or structures in the case of ambiguity) given in Lasnik & Uriagereka. In the case of ill-formed sentences or structures, the parser must report that the input violated the same principles as predicted in Lasnik & Uriagereka.

[4] On a tangential note, Chomsky [13] (also in [9] and earlier work) has argued because of the divergence between a (human) parser and grammar (with respect to performance issues), that there's no reason to expect grammars to be (readily) parsable. He also argues that an analogous observation holds with respect to learnability.

[5] Incidentally, by Fodor's definition, systems that maintain a strict separation of principles in visible definitions, but merge principles during a (hidden) compilation step would be non-modular. The system described in this thesis is modular provided no principles are interleaved.

1.4 Linguistic Coverage

As far as we are aware, the system implements a substantially larger (and more sophisticated) fragment of a principle-and-parameters theory than any extant parser.[6] To provide some idea of the capability of the implemented system, we will take the opportunity here to present examples of linguistic phenomena that the system can handle. To test the feasibility of implementing a substantial grammar, an introductory linguistics textbook, namely *A Course in GB Syntax* by Lasnik & Uriagereka [35], was chosen as a representative example of a current principle-based theory. The basic idea was to simply try to "implement" the book; that is, to implement the principles defined there, and to have the system correctly handle all the accompanying linguistic data. Of course, since the text covers several different (mutually incompatible) theories, it was not possible to achieve that goal. Moreover, the theory was found to be substantially incomplete. Consequently, additional principles were imported from *Knowledge of Language*, Chomsky [12], and other sources. Nevertheless, the subset of the principles implemented achieves substantial coverage. In fact, the system successfully accounts for almost all of the examples to be found in the first four chapters of the reference text.[7] For example, figure 1.1 contains a partial listing of representative sentences that the current system has been tested on. The surprising fact is that such coverage can be achieved using a core set of only twenty-five principles.[8]

As mentioned earlier, the parser produces correct parses in the sense that it makes the same judgements as described in the textbook for each example. That is, in the case when a sentence is "well-formed", the parser will return the structure, or structures, predicted by the theory. Operationally, a sentence will be considered to be *well-formed* only if there is at least one structural description that satisfies all the principles. On the other hand, if a sentence is *ill-formed*, there will be at least

[6] For example, consider Correa's attribute grammar formulation described in his thesis ([15]). The major differences in coverage can be attributed to the fact that his system is based on a much earlier instantiation of standard theory. Also, his parser lacks the Empty Category Principle (ECP), Control, LF-movement and (and the concomitant LF licensing constraints), head movement, adjunction resulting from movement, multiple-*wh*-extractions, genitive Case markers, and various other minor operations. Correa's system is essentially unparameterized and can only handle English. However, the system contains many interesting features which we return to in chapter 2.

[7] *A Course in GB Syntax* contains six chapters. Chapter five which covers extensions and alternatives to the Binding theory, Aoun's Generalized Binding and Higginbotham's Linking theory, has not been implemented. Chapter six discusses several open questions and unresolved problems. Of the various proposals discussed there, the current implementation has adopted the prohibition against Case-marking of NP-traces, and genitive Case realization (see Chomsky [12]) as a partial solution for *illicit NP-movement.*

[8] This figure does not include the grammar rules that define basic phrase structure syntax. There are ten basic $\overline{\text{X}}$-rules plus another twenty covering adjunction and the insertion of empty categories. A detailed description of the grammar rule system can be found in chapter 4.

(1:12a)	Someone likes everyone	*Quantifier raising*
(1:15a)	Who that John knows does he like	*Movement & Condition C*
(1:15b)	He likes everyone that John knows	
(1:18)	It is likely that John is here	*Simple Case theory*
(1:19)	*It is likely John to be here	
(1:21)	I am eager for John to be here	*Exceptional Case Marking*
(1:22)	*I am eager John to be here	*(ECM)*
(1:24a)	I believe John to be here	
(1:24b)	I believe John is here	
(1:25)	*I believe sincerely John to be here	*Case Adjacency*
(1:35a)	I want to be clever	*Optional vs. obligatory ECM*
(1:35b)	*I believe to be clever	
(1:36a)	John was persuaded to leave	*Ordinary passivization*
(1:48)	John was arrested by the police	
(1:36c)	*John was wanted to leave	*Exceptional passivization*
(1:49a)	I believe John to be intelligent	
(1:49b)	*It was believed John to be intelligent	
(1:49c)	John was believed to be intelligent	
(1:52a)	*I am proud John	*Genitive Case realization*
(1:52b)	I am proud of John	
(1:53a)	I wonder who you will see	Wh-*movement and Subjacency*
(1:59c)	*What does Bill wonder who saw	
(2:19b)	Their pictures of each other are nice	*Simple Binding theory*
(2:26a)	John likes Mary's pictures of him	
(2:26b)	*John likes Mary's pictures of himself	
(2:45a)	Who does he think Mary likes	*Strong crossover*
(2:88)	*(I am proud of) my belief to be intelligent	*Government of PRO*
(2:91)	The men think that pictures of each other will be on sale	
(2:103)	*The men think that Mary's pictures of each other will be on sale	
(3:17)	Which report did you file without reading	*Parasitic gaps*
(3:18)	*Which book did you file the report without reading	
(3:19)	*Who filed which report without reading	
(3:20)	*The report was filed without reading	✓*Resumptive pronoun*
(3:46)	*The article which I filed it yesterday without reading is (over) here	
(4:3)	*John is crucial to see this	*ECP*
(4:4)	John is certain to see this	
(4:7a)	Who do you think that John saw	*That-trace effect*
(4:7b)	Who do you think John saw	
(4:8a)	*Who do you think that saw Bill	
(4:8b)	Who do you think saw Bill	
(4:18)	*John is believed is intelligent	*Raising and Super-raising*
(4:20)	*John seems that it is likely to leave	
(4:21b)	Who will read what	*Superiority*
(4:21c)	*What will who read	
(4:35a)	Why did you read what	*Complement-noncomplement*
(4:35b)	*What did you read why	*asymmetries*
(4:45a)	Who believes the claim that Mary read what	*Subjacency and LF*
(4:45b)	*What do you believe the claim that Mary read	
(4:57)	*Who does Mary wonder why John hit	*ECP and syntactic movement*
(4:58)	*Why does Mary wonder who John hit	

Figure 1.1 Examples from Lasnik & Uriagereka

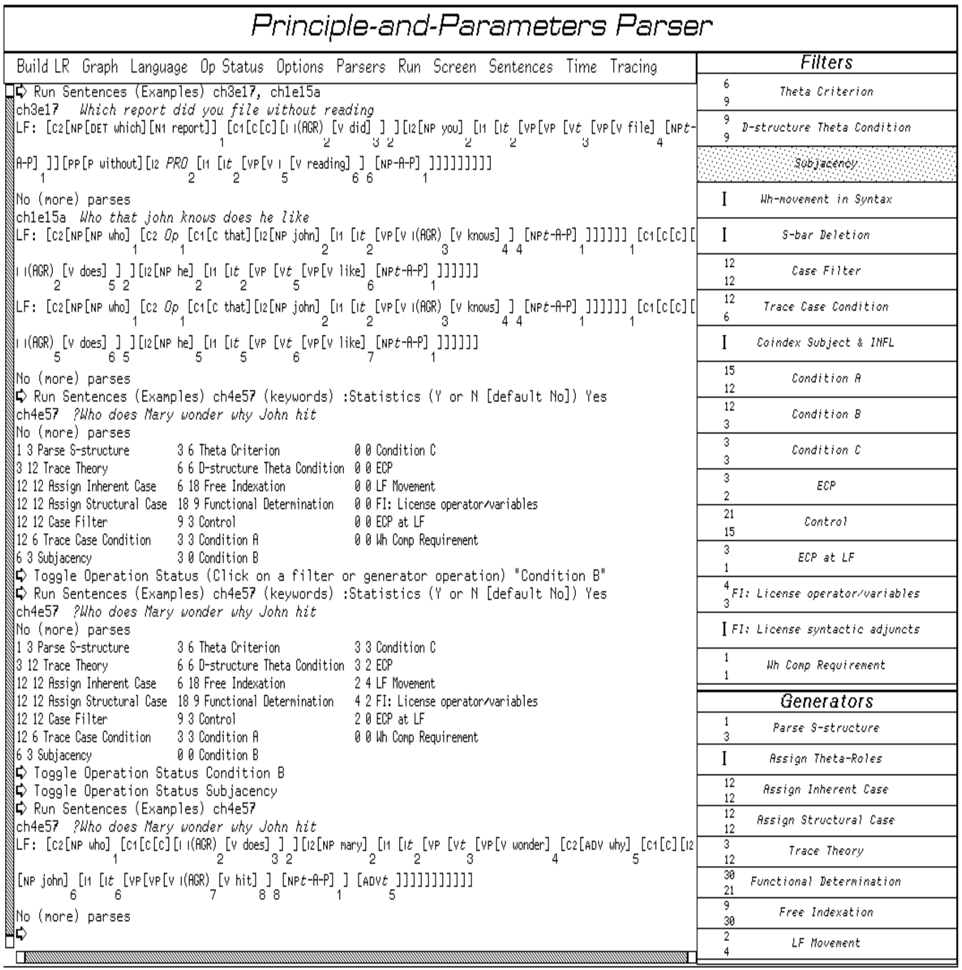


Figure 1.2 System snapshot.

one principle that no structural description will satisfy.

Consider the system snapshot shown in figure 1.2. The display is divided into two parts. Parses are displayed on the left. The right side contains a panel listing the parser operations. In most cases, each parser operation will map onto a single principle. This panel is used to provide an animated display that provides much useful information on how parsing is proceeding. Basically, there are two counters associated with each operation. (Some operations may have an 'I' instead of two counters. The relevance of this mark will be explained when we return to discuss parser control.) The upper counter records how many times the operation was called, and the lower counter records how many times the operation succeeded. The relative number of calls and successes will indicate if a principle is behaving as a *filter*, i.e. ruling out structures, or as a *generator*, i.e. instantiates more structures than it eliminates. For example, the operation *Parse S-structure* has been called once, but has succeeded three times, which means that there are three possible phrase structures that can be assigned to the input sentence at S-structure. Similarly, *Condition B* has ruled out nine of the twelve structures passed to it.[9] We can also identify the most and least active operations in terms of eliminating ill-formed structures. For example, unlike *Condition B*, the *Case Filter* has had no effect since all of the structures it has received have been allowed to pass. Generally, if a sentence is well-formed, all counters will be non-zero. When a sentence is ill-formed, there will be a single operation F (the current system executes operations serially) for which the number of calls will be non-zero but there will be no successes registered. Then, let us say for now that the sentence is an example of a F filter violation. (We will revise this definition shortly.)

The display on the left contains the result of parsing three sentences. Let us consider each of them in turn:

- *Identifying coreference relations.*

  The first parse at the top of the figure shows the LF structure returned for the *parasitic gap* sentence *which report did you file without reading?* (1) (= (3:18) in Lasnik & Uriagereka) that was discussed at the beginning of the chapter.

  The LF shown here has considerably more detail than necessary to identify the coreference relations. We will just pick out the relevant information. The object positions immediately to the right of both *file* and *reading* have been occupied by two empty noun phrases, shown as [NP*t*-A-P] and [NP-A-P], respectively. In either case, the features -A (non-anaphoric) and -P (non-pronominal) indicate that they are not subject to either Binding condition introduced earlier in the chapter. The fact that they both have index 1, as indicated by the subscripts shown, will mean that they will be both interpreted as variables bound by the NP *which report* that shares the same index.[10] Using the simple interpretation rule:

[9] Actually, most of the structures passed around are merely further instantiations of the same basic structures. For example, all twelve of the structures produced by *Trace Theory* actually are variants of the three (underspecified) phrase structure generated by *Parse S-structure*. The twelve structures are simply distinguished by different assignments of movement chain features. In particular, no constituents have been discarded or built by *Trace Theory*.

[10] Note that there is a syntax-internal difference (irrelevant to interpretation) in the derivation of the two variables. [NP *t*-A-P], as indicated by the '*t*' is the trace of *which report*, i.e. it is formed as a result of fronting *which report*. On the other hand, [NP-A-P] is not formed by movement, but "base-generated".

(5) Two elements are coreferential if and only if they share the same index

and ignoring all indices not attached to NPs, we can see that the parse contains the correct coreference option. In a similar fashion, the embedded subject position of the prepositional phrase *without reading* has been occupied by an empty NP, *PRO*. As expected, *you*, the subject of *file*, gets the same index as *PRO*, namely 2 and, crucially, not 1.

- *Reporting ambiguity.*

  In the case of ambiguity, the system will return as many structures as the degree of ambiguity demands. For example, consider the sentence:

  (6) who that John knows does he like?

  This is the next sentence shown in the snapshot. Here, the two possible LFs returned for are identical save for the index on *he*. The first LF corresponds to the case where *he* should be interpreted as being coreferential with *John*, the second LF corresponds to the case where *he* refers to someone not named in the sentence (ignoring non-NPs again, index **5** does not occur anywhere else in the structure).

- *Explaining ill-formed input.*

  Consider the following ill-formed sentence:

  (7) *who does Mary wonder why John hit? = *(4:57) in [35]*

  According to Lasnik & Uriagereka (section 4.4.2), this is a straightforward example of a *Subjacency* violation. The basic assumption here is that *who* has been extracted from its original position as the object of *hit*. The standard explanation why the sentence is ill-formed is that *who* has moved "too far" from its base position. Now consider the output shown in the snapshot. No LF has been returned since the sentence is ill-formed. Instead, the parser has reported the counter values for each operation.[11] Surprisingly, the "blocking" filter reported in this case is *Condition B* from the Binding theory, not *Subjacency* as predicted by Lasnik & Uriagereka.

  A useful feature of the system is that it can be directed to turn off any parser operation. Thus, ill-formed parses can sometimes be "saved" by simply turning off the relevant blocking filters. For some cases, this will result in a still

[11] The order in which the operations have been executed can be determined by reading the list top-down, starting from the leftmost column.

interpretable sentence.[12] So, let us explore what happens when *Condition B* is turned off. (This is accomplished by the 'Toggle Operation Status' command.) Running the sentence again, we see that it is still blocked, but this time at the *Empty Category Principle (ECP)* that operates at LF. Therefore the sentence cannot be an example of a *Condition B* violation.

Let us now backtrack and see if Lasnik & Uriagereka were correct. First, we need to re-enable *Condition B* and then turn off *Subjacency*. (This is indicated in the latter case by the "blanked out" display on the right panel.) Running the sentence one more time, we can see that a LF is returned. Again, ignoring irrelevant details in the structure, both *who* and the EC [NP*t*-A-P] immediately following the verb *hit* have the same index. Moreover, the '*t*' indicates that it is indeed the trace of movement of *who*, as Lasnik & Uriagereka predicted. Therefore, the sentence is not a *Condition B* but a *Subjacency* violation. Hence, we should revise the tentative definition of a F filter violation to include the condition that the sentence is well-formed if F does not apply.[13]

In general, the search for all possible parses can produce somewhat unexpected results. In the next section, we will illustrate other cases when the parser actually produces unanticipated parses. We note that, in such cases, the parser is not "wrong" in the sense that it is producing parses that are inconsistent with linguistic theory, but rather that the theory as implemented is under-constrained.

1.4.1 Language Parameters

Using basically the same set of principles, but with a different language parameter vector and lexicon, the system can be automatically re-configured to parse Japanese examples instead. We would like to emphasize, however, that the system has not been rigorously tested on Japanese sentences. So far, only a very limited set of test cases have been tried, namely the Japanese sentences found in the linguistics paper *On the Nature of Proper Government* by Lasnik & Saito [34]. These sentences are listed in figure 1.3. (Note that they are all *wh*-questions.) Nonetheless, these sentences display many of the typological Japanese–English differences. Let us now consider some of these:

[12] This roughly corresponds to the situation in which a human parser detects that a sentence is "syntactically" anomalous, but nevertheless is still able to assign it an interpretation.

[13] However, one mystery remains: What were the structures that got past *Subjacency* but *Condition B* ruled out? All three were of the form: *who does Mary wonder why* $John_i$ *hit* PRO_i. To satisfy the θ-criterion, both *who* and *PRO* are assumed to have been base-generated in Comp and object position of *hit*, respectively. *PRO* had to be bound by *John* in order to satisfy *Condition A*. But, with *PRO* bound in its governing category, all three are subsequently ruled out by *Condition B*.

(2)	Watashi-wa Taro-ga nani-o katta ka shitte iru *(I know what John bought)*	*Basic* wh-*questions*
(6)	Kimi-wa dare-ni Taro-ga naze kubi-ni natta tte itta no *(To whom did you say that John was fired why)*	*Good in Japanese* *but not in English*
(32)	*Meari-wa Taro-ga nani-o katta ka do ka shiranai *(Mary does not know whether or not John bought what)*	*Semantic parallelism:* *non-absorption of* ka do ka
(37a)	Taro-wa naze kubi-ni natta no *(Why was John fired)*	*Comp-to-Comp* *movement at LF*
(37b)	Biru-wa Taro-ga naze kubi-ni natta tte itta no *(Why did Bill say that John was fired)*	*Long distance* *movement of* naze
(39a)	Taro-ga nani-o te-ni ireta koto-o sonnani okotteru no *(What are you so angry about the fact that Taro obtained)*	*Complement-noncomplement* *asymmetries*
(39b)	*Taro-ga naze sore-o te-ni ireta koto-o sonnani okotteru no *(Why are you so angry about the fact that Taro obtained it)*	
(41a)	Hanoko-ga Taro-ga nani-o te-ni ireta tte itta koto-o sonnani okotteru no *(What are you so angry about the fact that Hanoko said that Taro obtained)*	
(41b)	*Hanoko-ga Taro-ga naze sore-o te-ni ireta tte itta koto-o sonnani okotteru no *(Why are you so angry about the fact that Hanoko said that Taro obtained it)*	
(60)	Kimi-wa nani-o doko-de katta no *(Where did you buy what)*	*Multiple-*whs *in Comp*
(63)	Kimi-wa nani-o sagashiteru no *(Why are you looking for what)*	

Figure 1.3 *Wh*-movement examples in Japanese from Lasnik & Saito.

- *SOV Language.*

  Japanese is classified as a SOV (Subject-Object-Verb) language. In general, heads such as verbs and adjectives are preceded by their objects and modifiers. However, subjects do normally appear before verbs and objects, as in English. This distinction can be encoded by two binary parameters that specify head/complement and specifier/head order: [14]

Language	Head/ Complement	Specifier/ Head
English	head first	specifier first
Japanese	head final	specifier first

- *Scrambling.*

  Apart from the fact that sentences normally begin with a topic and end with a verb, the order of other elements in the sentence is relatively free. In partic-

[14] We have oversimplified things here somewhat. The parser can use a slightly more complex system. In general, the actual order can vary depending on the syntactic category. See chapter 3 of Webelhuth [57] for a summary of the differences between the seven modern Germanic languages.

ular, direct and indirect objects can be switched, direct (and indirect) objects can appear in front of the subject in sentence-initial position. For example, consider the following four sentences corresponding to *John gave Mary a book* (taken from Hoji [29]):

(8) a. [*S* John-ga [*VP* Mary-ni hon-o ageta]]
b. hon-o John-ga Mary-ni ageta
c. Mary-ni John-ga hon-o ageta
d. John-ga hon-o Mary-ni ageta

This can be accomodated in the present theory using movement. The canonical order is assumed to be 'subject' followed by 'indirect object' followed by 'direct object', as in (8a). Then, the direct object, *hon-o* in this case, is free to move (by *VP-adjunction*) to a position in front of the indirect object *Mary-ni*, as in (8d), or to a sentence-inital position (by *S-adjunction*) as in (8b). Similarly, the indirect object may move to a sentence-initial position as in (8c).

- *Empty subjects.*

  Subjects can be omitted in Japanese. (In general the conditions that determine which elements can or can not be omitted are largely dependent on discourse considerations. We ignore such concerns here.) For example, consider the second sentence in (9) (taken from Makino & Tsutsui [38]):

  (9) Taro-wa sono mise-de nani-o kaimashita ka.
  (What did Taro buy at the store?)
  Pen-o kaimashita
  (He bought a pen)

  Other languages with this feature include Chinese, Italian and Spanish. In standard theory, the omitted subject is actually represented in syntax by an empty pronoun *pro* (which is not available in English). The binary option that determines whether *pro* is available or not is known as the *pro*-drop parameter.

- *No visible Wh-movement.*

  In English, wh-words such as *what* and *why* must appear in clause-initial position, as shown in (10) (taken from Lasnik & Saito [34]):[15]

  (10) a. I know [what John bought]
  b. *I know John bought what

[15] Excluding cases of echo-questions such as *you saw who*, and sentences containing multiple-*wh*-words — for example, see sentences (4:35a) and (4:45a) in figure 1.1.

It is usually assumed that (10b) is the "underlying", or *D-structure*, form for (10a). The *wh*-noun phrase *what* then moves to a position at the front of the embedded clause. By contrast, in Japanese the correspond *wh*-word, *nani*, remains in object position, as example (11) illustrates (= (2) in figure 1.3):

(11) Watashi-wa Taro-ga nani-o katta ka shitte iru

Although, Japanese sentences do not seem to exhibit *Wh*-movement as in English; it is generally assumed that similar sentences will have basically the same representation at LF. As with *which report* in the parasitic gap sentence discussed at the beginning of the chapter, it is assumed that *what* behaves somewhat like a quantifier in logic. In particular, it must move to a position that has clausal scope. To summarize, the basic difference here is that *wh*-elements move at different levels of representation in English and Japanese. That is, the following generative model of phrase structure is assumed:

$$\begin{tikzcd}[column sep=small] & \text{D-structure} \arrow[d] & \\ & \text{S-structure} \arrow[dl] \arrow[dr] & \\ \text{Phonetic Form} & & \text{LF} \end{tikzcd}$$

Here, roughly speaking, *Phonetic Form* serves as the level of input for parsing. Hence in Japanese, there wil be no *wh*-movement between D- and S-structure; all cases of *wh*-movement taking place "invisibly" between S-structure and LF. Thus, the option of whether to allow *wh*-movement between D- and S-structure is a language parameter. We will not comment further except to note that this distinction has enabled Lasnik & Saito to explain a variety of facts including why the counterpart of a sentence such as (6) in figure 1.3 (which is well-formed in Japanese) is ill-formed in English.

While obviously this is very far from being a complete characterization of the differences between Japanese and English, it is sufficient to cover the examples shown in figure 1.3. Figure 1.4 shows the system in operation for the first five sentences from figure 1.3. We will just point out a few interesting features of some of these parses:

- Wh-*movement at LF.*

  Consider example (11) that was discussed above. As shown at the top of the snapshot, *nani* has moved at LF to a position that has scope over the embedded sentence (as indicated by the bracketing) leaving behind a trace LFt, to be interpreted as a variable, in its original position.

- *Multiple wh-elements at LF.*

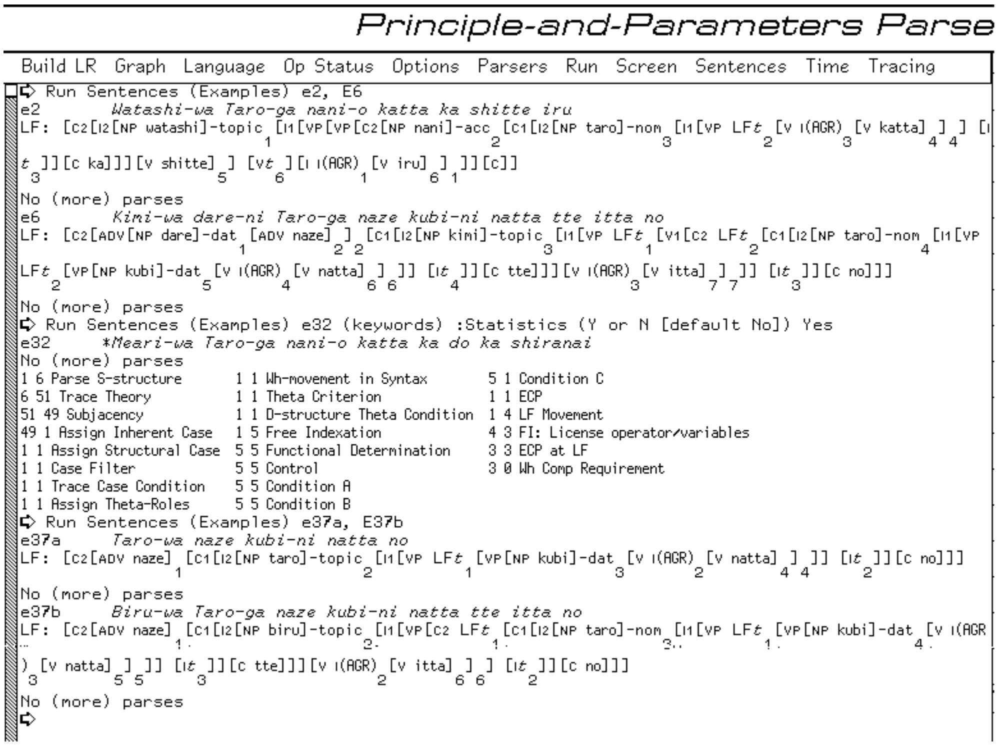


Figure 1.4 Examples from Lasnik & Saito: Part 1.

Consider example (6) in figure 1.3. There are two wh-elements: (1) *dare* ('who') the indirect object of *itta* ('said' as in 'said something to somebody'), and (2) *naze* ('why'). As the second parse shows, both elements will move at LF to a sentence-initial position. Hence, roughly speaking, the sentence will be interpreted as a double question: 'for which x, x a person, *and* for what reason y, you said y (a reason such that [taro was fired]) to x.'

We will conclude this section with a demonstration of how the parser can produce unexpected parses. Consider (39a) from figure 1.3, repeated here as (12):

(12) Taro-ga nani-o te-ni ireta koto-o sonnani okotteru no
(What are you so angry about the fact that Taro obtained)

(Here, note that the subject of the matrix clause (= *you*) has been omitted. Also, *nani* and *te* ('hand') have been permuted from the canonical order described above — a simple case of scrambling.) The LF for this sentence should be something along the lines of:

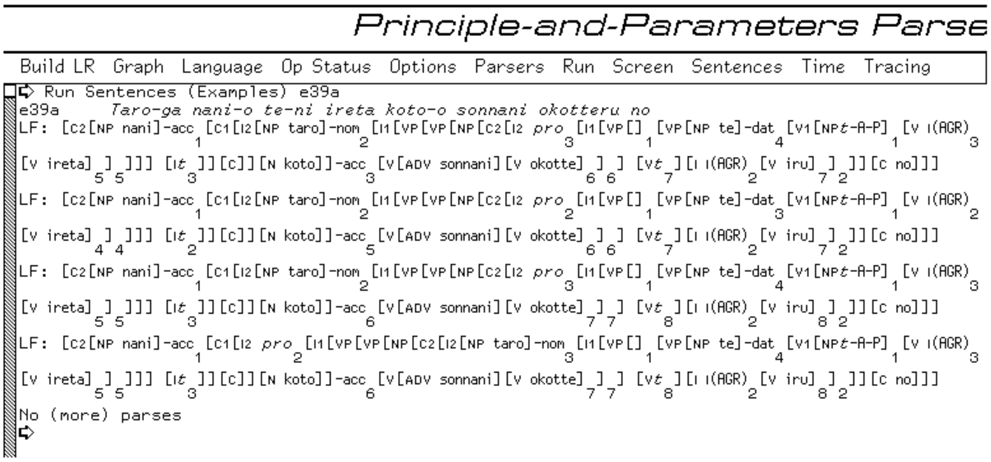


Figure 1.5 Examples from Lasnik & Saito: Part 2.

(13) for what x, *pro* is so angry about [the fact that Taro obtained x]

Here, *pro* represents the understood subject of *okotteru* ('be angry'). Now consider the LFs returned by the system in figure 1.5. As we can see, the system does correctly recover this form, as the last LF in the snapshot. However, it also recovers three additional LFs which (ignoring indices for the time being) all have the same form:

(14) for what x, Taro is so angry about [the fact that *pro* obtained x]

The essential difference is that the parser has predicted that it is possible to "exchange" the embedded subject *Taro* for the matrix subject *pro*.[16] As it turns out, the basic prediction is correct as (12) happens to be ambiguous with respect to the two basic interpretations.[17] (Actually, the original example as stated in [34] is unambiguous because of the use of bracketing.[18]) This concludes the overview of the linguistic capabilities of the system. In the next section, we will briefly describe the available control options.

[16] For simplicity, we have omitted discussion of many complex but interesting details about the parses returned by the system. The actual derivation sequence in figure 1.5 is quite complicated. For example, the observant reader may notice that *nani* ('what') undergoes both S-structure and LF movement. At S-structure, the direct object "scrambles" by VP-adjunction leaving the trace $[\mathrm{NP}t\text{-A-P}]_1$. At LF, *nani* undergoes *wh*-movement into COMP (actually, the specifier position of COMP) with the LF trace being subsequently deleted, as indicated by $[]_1$.

[17] This was pointed out to the author by D. Pesetsky, and confirmed by M. Saito.

[18] For completeness, here are the three variants of (14) that correspond to the first three LFs reported by the system:

1. *pro* is coreferent with *koto* ('fact'):

1.5 Parser Control

In the previous section, we have seen one simple example where the system allowed the user to control how principles are applied. There we described how filter violations could be verified by turning off certain parser operations. In this section, we will turn our attention to control options of a different nature. These control options have the common property that all linguistic judgements are preserved. Therefore, the only difference that the user will see is in the time it takes for the system to return parses.

Most existing principle-based parsers have their program control "hardwired" into the system. For example, the order in which principles are processed is normally fixed. However, a natural question arises: What is the justification for preferring any one ordering over another? Or more generally we can ask: What is the most efficient way to apply principles? For some systems such as Abney's [2], the goal has not been necessarily to find the most effective organization, but instead to find one that is psycholinguistically plausible. Since the linguistic theory given in Lasnik & Uriagereka is one that deals with aspects of language competence rather than performance, the system described here has no processing constraints of that sort; the task being simply to recover all possible parses in no particular order.

In general, it may not be possible to predict an appropriate control strategy ahead of time, i.e. without doing some experimentation. Furthermore, an appropriate strategy may not continue to be one if principles are altered. To conclude, there seems to be little reason in the current system to unnecessarily restrict the range of control options.[19] Although it would be quite infeasible to cover all possible parsing strategies, the approach taken here is to allow simple variations on the basic processing framework, so long as the system is able to re-configure the parser control *automatically*, i.e. without assistance or special programming on the part

---

for what x, Taro is so angry about [the fact that the fact obtained x]

This interpretation can probably be eliminated by imposing selectional restrictions on the possible 'agents' of *okotteru*.

2. *pro* is coreferent with *taro*:

   for what x, Taro is so angry about [the fact that Taro obtained x]

   (We have not consulted a native Japanese speaker on the grammaticality of this interpretation, but currently the system does not rule it out.)

3. *pro* is free in the sentence:

   for what x, Taro is so angry about [the fact that (someone else) obtained x]

[19] Perhaps being able to vary control parameters (even if those parameters may not quite be the appropriate ones) would be a useful option for parsers that attempt to model psycholinguistic predictions.

of the grammar writer, and *transparently*, i.e. without missing well-formed parses, accepting ill-formed sentences, or altering the structures that the user sees in any way. (We will return to consider possible strategies in more detail in chapter 2.)

1.5.1 An Outline of the Section

This section consists of three parts:

1. In the first part, we will introduce the two models of parsing that the present system accommodates. We will discuss the *principle ordering* model of parsing that relies on the fact that principles are generally unordered. Next, we describe an alternative *principle interleaving* model in which principles are applied to partially-built structures as phrase structure construction proceeds.

2. We will illustrate the differences in parsing times that can occur as a result of different control choices with a simple comparison of two parsers, one based on the principle ordering model and the other one a hybrid model that combines both approaches.

3. We will end the section with a general overview of the control decisions and tradeoffs for configuring efficient parsers in the present system that will be discussed in depth later in chapters 5 and 6.

1.5.2 Two Models of Parsing

- *Principle ordering.*

  Consider again the panel of parser operations shown in the system snapshot of figure 1.2. The basic model of parsing allowed by the system is to connect, or "link", up these operations in some prescribed order, and then to pass hypothesized structures from one operation to the next in an assembly line fashion. Each operation may: (1) reject or pass on without comment the structure it is given, thus behaving as a filter, or (2) augment the structure in some way, e.g. the operation *Assign Theta-Roles* instantiates the θ-slots of the arguments of heads such as verbs, nouns and adjectives, or (3) create another level of representation by rearranging the elements of the existing structure, e.g. the *LF movement* operation may move *wh*-noun phrases and quantifiers to clause-initial positions. In general, the user is free to rearrange the "assembly line" order, say, to obtain faster processing (as we shall see below). Hence, a *parser specification* in this system is simply a description of how the operations are to be connected.

  However, there are certain restrictions on what permutations are allowed. (See the dependency graph in figure 1.6.) For example, the system does not

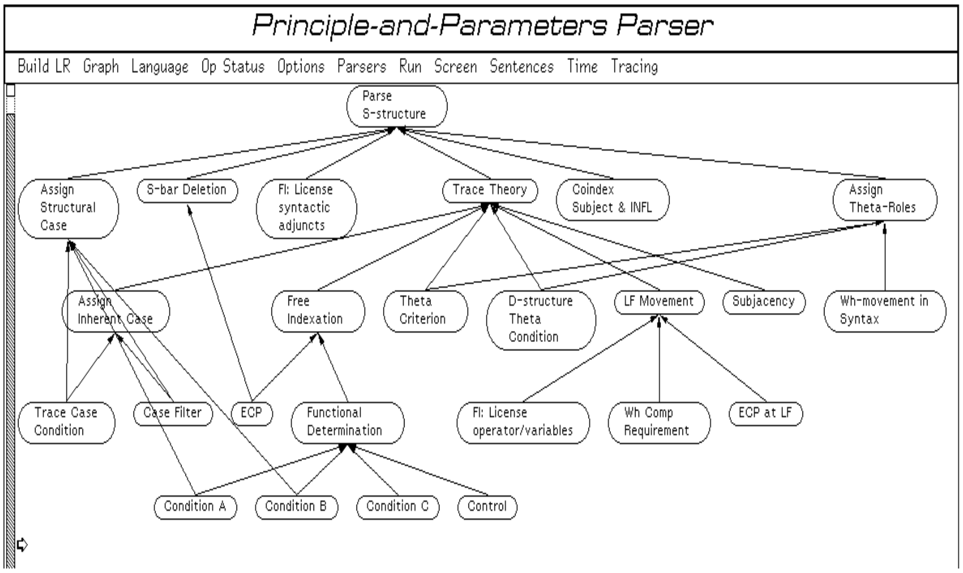


Figure 1.6 Operational dependency graph.

allow the construction of a parser that puts the *Case Filter* (a requirement that all non-empty noun phrases be assigned Case) ahead of either *Inherent Case Assignment* or *Structural Case Assignment* (the two operations that can assign Case to noun phrases in certain configurations). In turn, neither Case assignment operations can be put ahead of *Parse S-structure* (the operation that builds syntax trees) because both operations need to know the positions that noun phrases occupy in syntactic structure in order to know which noun phrases are permitted to receive Case. (Note that this is an overly simplistic model; that is, the restrictions described here are not fundamental to the linguistic theory, but only to this particular implementation.[20])

- *Principle interleaving.*

  The "unit of currency" adopted in the assembly line model is a "complete" phrase structure tree. (Here, by "complete" we mean only that all constituents

[20] For example, we can remove such restrictions by viewing the Case assignment process in a slightly different way. We can re-express the *Case Filter* as simply marking each overt NP with a feature indicating that the NP must get Case. Then we can proceed with both Case assignment operations in the normal fashion. Finally, we can check that each *Case Filter* tag was properly satisfied. Another option is to eliminate this extra last step by combining the three Case theory operations into one that visits each noun phrase in turn. However, for simplicity and other design considerations that we will discuss later, we do not consider these alternate formulations.

have been constructed. In particular, syntactic features of individual constituents, e.g. Binding features [±A,±P] or indices, may remain incompletely instantiated.) One limitation of this model is that principles cannot be applied to smaller fragments of phrase structure. One reason why one might want such a model would be to avoid the unnecessary work of constructing a complete but ill-formed phrase structure if some fragment of that structure could be shown to be ill-formed independently from the rest of the structure. For example, consider the sentence:

(15) *[the man I saw him] wondered why Mary bought the tree

The matrix subject here is the (ill-formed) relative clause construction *the man I saw him*. If the parser can apply the principle that rules out this matrix subject without building the structure for the whole sentence, then we can save a considerable amount of work. The present system accommodates such strategies by allowing the user to specify that principles should be *interleaved*, i.e. co-routined, with phrase structure construction in *Parse S-structure*. Then, *Parse S-structure* will only output complete structures that satisfy the interleaved principles. (We will return to discuss the connection between relative clauses and principle interleaving in the context of *licensing parsers* in chapter 2.)

Note that the two models we have described are not mutually exclusive options. In the current system, a parser specification consists of two parts: (1) a designated subset of principles to be interleaved (possibly none), and (2) a particular ordering of the remaining principles. (We emphasize that such control options are specified independently from the representation of the principles used by the system. It is perhaps also worth pointing out that the grammar writer need not be concerned about how principles are to be interleaved (or re-ordered). Both procedures are carried out in a completely automatic fashion by the system.)

1.5.3 Two Parsers, One Theory

To illustrate the difference in parsing time that can result from different parser specifications using exactly the same set of principles, consider the system snapshot shown in figure 1.7.

Here, the results of parsing *who does Mary wonder why John hit* (= (7), the subjacency violation discussed in the previous section) are shown for two different parsers. The default parser used by the system is one in which no principles are interleaved. Instead, the principles are applied in the order given by the list of operations at the top left side of the snapshot. Here, the order of application can be read top-down starting at the left-most column. So *Parse S-structure* is applied

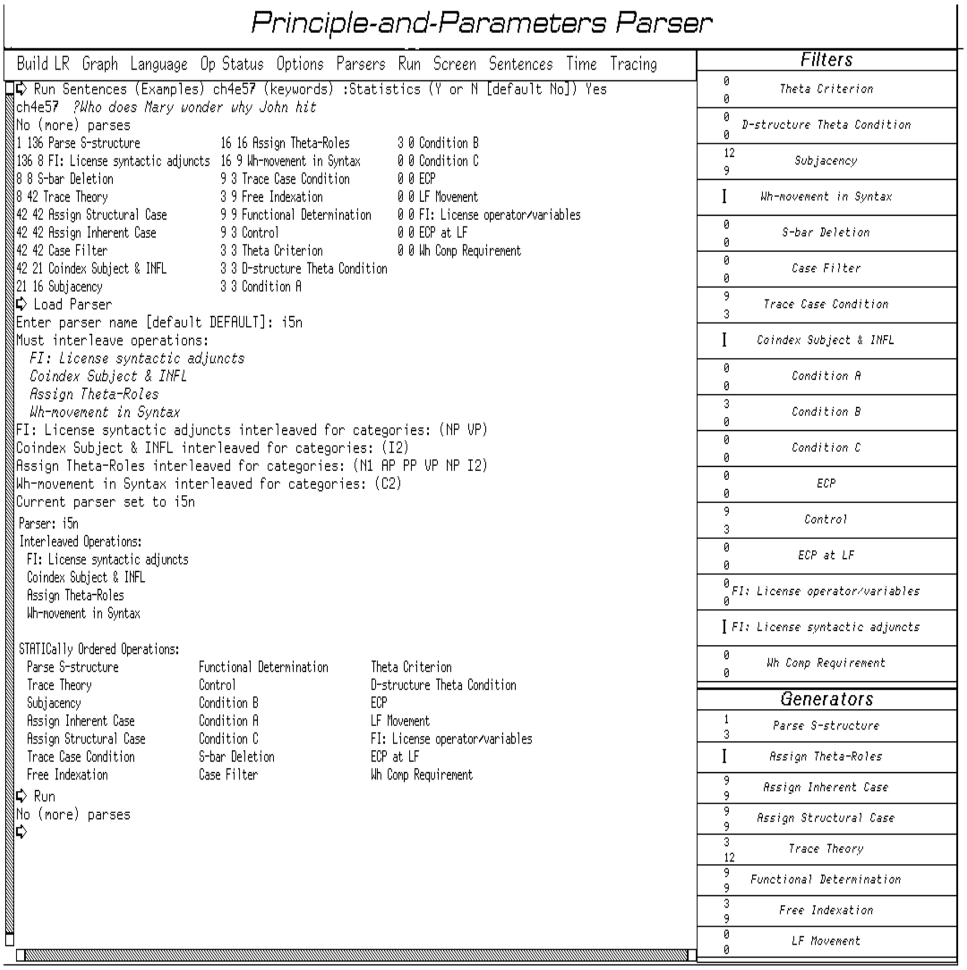


Figure 1.7 Two parsers with different control strategies.

first, followed by *License Syntactic Adjuncts*, then *Trace Theory*, and so on. (Note that the number of calls recorded for each operation is the same as the number of successes recorded for the immediately preceding operation in the chain.) As the record of calls and successes indicates, nineteen out of the twenty-five operations were invoked in the course of rejecting the input sentence.

Next, the snapshot shows the loading of a different parser, i5n. (We will not discuss the significance of the lists of categories shown for each interleaved operation here. A complete explanation can be found in chapter 6.) Here, we only note that the system has been re-configured with four operations interleaved, and the remaining twenty ordered quite differently from the previous configuration. For example, that *Condition B* now appears in front of *Condition A*, *Subjacency* has been moved up four places ahead, and the *Case Filter* has been relegated to the second column. Running the same sentence again, we can see (from the panel on the right side) that only nine operations (compared with nineteen) were consulted this time. (No calls or successes have been recorded for the four interleaved operations (each marked with an 'I' in the right panel) because they have been completely integrated into the *Parse S-structure* operation.) In fact, parser i5n caused parser operations to be invoked a total of 57 times — a considerably smaller number than for the previous case (a total of 415 calls).[21] Using total time taken as a measure of relative efficiency, i5n ran almost nine times faster than the default parser.

This is not a surprising result. Of the nineteen operations invoked by the first parser, only five were useful in the sense that they helped to eliminate ill-formed structures. Most of the remaining fourteen could be eliminated without affecting the overall linguistic judgement since they did no useful work. For example, the *Case Filter* operation eliminated none of the forty-two structures it was passed. In i5n, the *Case Filter* appeared in a position later than the blocking filter, so it was never called. Of the five effective filters, note that three were interleaved in i5n so that they could be applied as early as possible. Moreover, parser i5n also included no operation in a position ahead of the blocking filter that was not either a useful filter, or an operation that was required as a precondition by the dependency restrictions in figure 1.6.

1.5.4 Parser Control for Efficiency

In the following paragraphs, we will review the issues involved in choosing an efficient control configuration. In particular, we will discuss the differences and trade-offs between *dynamic* and *static* ordering, factors that affect the utility of inter-

[21] One caveat: it would be dangerous in general to extrapolate directly from the relative number of calls to the relative time actually spent parsing. In particular, the time spent per call on *Parse S-structure* cannot be directly compared. The fact that four operations have been merged into *Parse S-structure* will affect the unit time per call.

leaving, and the effect of *well-formed* and *ill-formed* input. We will also take the opportunity to summarize results and observations (to be covered in depth in chapters 5 and 6) derived from tests on a large number of examples from Lasnik & Uriagereka.

Principle ordering

- *The filter and generator model.*

  In general, it will prove more efficient to apply the simple strategy of ordering principles so that "filters" and "generators" (in the operational sense defined above) are applied as early and as late as possible (provided data dependencies between principles are respected), respectively. The essential idea being to eliminate hypothesized structures as early as possible. By ordering filters in terms of their relative *filtering power*, i.e. the proportion of structures eliminated, it is often possible to improve upon the default configuration by an order of magnitude or more.

- *Hardwiring a specific ordering.*

  Unfortunately, as will be discussed in chapter 5, factor-of-ten improvements over the default configuration are difficult to maintain. A particular configuration such as i5n may be much more effective for certain examples such as (7), but may be only slightly faster or even slower for others. The basic reason is that different inputs will "exercise" different parser operations. For example, the Case Filter which did no work in the example in the previous section figures prominently as the dominant filter for examples such as **I am eager John to win — cf. I am eager for John to win.*

- *Dynamic ordering.*

  Although, it does not seem to pay to hardwire any one parser configuration; on the other hand, the possibility of a factor-of-ten improvement does suggest it may be worthwhile to consider "on-the-fly" reconfiguration. This leads us to briefly mention one other control option. As chapter 5 will describe, another possibility is to allow the parser to *dynamically* pick its own ordering for each structure generated on the basis of the presence of certain "cues" in a structure instead of imposing a *static* ordering as we have seen here. The key observation here being that most structures generated by the parser will be ill-formed. By examining structural cues, the dynamic ordering mechanism tries to "approximate" the filters by guessing which filter each structure will violate *without* incurring the expense of actually applying the filters.

  The cost-benefit tradeoff that determines the success of dynamic ordering depends on a combination of how reliable structural cues are, and the amount of overhead, both in terms of the cost of making those filter predictions and the extra machinery required for re-organizing the assembly line order. In the current implementation, the goal has been to try to minimize this overhead as much as possible by only making use of cues that require relatively little computation to determine, e.g. testing for the presence of certain syntactic features. However, in general there will be a tradeoff in terms of the simplicity of the approximation, i.e. the cost of computing the approximation, and the reliability of the resulting predictions. Finally, we should also point out that eliminating ill-formed structures is almost as important in parsing well-formed as well as ill-formed sentences. The reason being that, even for well-formed sentences, almost all of the structures hypothesized by the parser must be ill-formed, unless, of course, the input sentence is massively ambiguous.

Principle interleaving

- *On-line interleaving is impractical.*

  As chapter 6 will explain, it will not prove effective to interleave parser operations at run-time because of the overhead involved. More precisely, a typical parser operation has a very limited "range" of application in the sense that many pre-conditions may need to be satisfied before, say, some feature is assigned or some condition is applied. For example, before Case can be assigned: (1) a Case assigner and a noun phrase to receive Case must be found, (2) the assigner must "govern" the receiver which holds only in a certain number of configurations, and (3) the assigner must be adjacent to the receiver (at least in English). In fact, most of the work in applying Case assignment comes from verifying that the above pre-conditions are met. A parser that tries to apply Case assignment to partial phrase structures as soon as possible will spend a considerable amount of effort failing to assign Case.

- *Eliminating the overheads of interleaving.*

  However, if the principle to be interleaved is known beforehand, then a considerable amount of unnecessary testing can be eliminated by using an off-line type inference procedure (described in chapter 6) to identify syntactic configurations that can never be configurations of Case assignment.

  The utility of the interleaving mechanism also depends to a large extent on seemingly unrelated factors such as the relative "efficiency" of the phrase structure recovery mechanism. The pertinent notion of efficiency in this case will be the ability of the parser to detect "dead-ends" in structure recovery (i.e. the fact that the current sentential form cannot be reduced to a single sentential symbol), as early as possible. The current implementation uses a canonical LR(1)-based algorithm (described in chapter 4) to generate phrase structures. It is well-known that canonical LR(k) parsers are guaranteed to detect erroneous sentential forms as early as possible given k symbols of terminal lookahead. However, one surprising result of empirical testing is that, even with a low intrinsic overhead, it is more efficient *not* to interleave for many parser operations.

This concludes our discussion of the control options of the system. In the next section, we will turn to discuss implementation issues.

1.6 The Implementation: Grammar Representation and Compilation

Ihis section will briefly review how the grammar, i.e. the phrase structure rules and principles, is encoded and subsequently turned into parser operations. The section is divided in three parts:

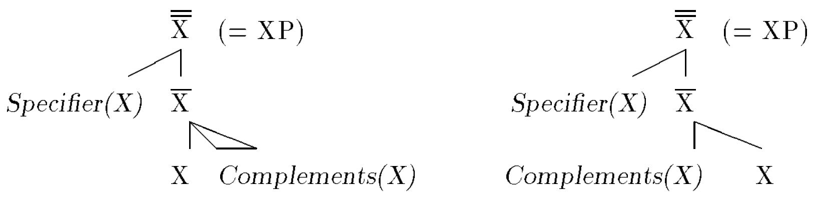
1. The representation of phrase structure rules.
2. The representation of principles using various primitives defined on constituent structure.
3. The organization of the various sub-systems that turn grammar and control definitions into parsers.

1.6.1 Phrase Structure Rules

This section provides a brief overview of the representation used for the phrase structure component of grammar. Chapter 4 contains a complete description of the representation used and its compilation into canonical LR(1)-based tables.

$\overline{\text{X}}$-theory

In $\overline{\text{X}}$-theory, all phrases have the same basic internal structure. Two forms of a typical $\overline{\text{X}}$-prototype for a generic phrase are shown below:



(English) (Japanese)

As discussed earlier in section 1.4.1, Japanese is classified as a SOV language and thus requires a different order of phrasal components from English. In either case, X ranges over the usual categories N (noun), V (verb), A (adjective) and P (preposition), with extensions to include sentences (S) and full clauses ($\overline{\mathrm{S}}$). The lowest and highest levels corresponds to the traditional levels of individual lexical items and complete phrases, respectively.

In the current system, a small number of non-ground rules headed by the following two rules is used to represent the $\overline{\mathrm{X}}$-prototype:

```
rule XP -> [X1|specifiers(X1)] ordered specFinal
                                 st max(XP), proj(X1,XP).

rule X1 -> [X|complements(X)] ordered headInitial
                                st bar(X1), proj(X,X1).
```

Some PROLOG notation: logical variables are distinguished from constants by the use of an uppercase letter as the initial character. The scope of a variable is a clause.

Elements delimited by square brackets denote lists. In particular, `[]` denotes the empty list. `[Y]` denotes a list with one element `Y`, `[Y1,Y2]` a list of two elements, and so on. `[X|L]` denotes a list with head `X` and tail `L`.

Here, `specFinal` and `headInitial` are binary language parameters that determine the relative order of the right-hand-side elements. The possible values for `XP`, `X1` and `X` are defined by the predicates `max/1`, `bar/1` and `head/1` as follows:

% Noun	*Verb*	*Adj.*	*Prep.*	*S*=`i2`	$\overline{S}$=`c2`	*Level*
`head(n).`	`head(v).`	`head(a).`	`head(p).`	`head(i).`	`head(c).`	% X
`bar(n1).`	`bar(v1).`	`bar(a1).`	`bar(p1).`	`bar(i1).`	`bar(c1).`	% $\overline{\mathrm{X}}$
`max(np).`	`max(vp).`	`max(ap).`	`max(pp).`	`max(i2).`	`max(c2).`	% $\overline{\overline{\mathrm{X}}}$

```
proj(X,Y) :- head(X), bar(Y).
proj(X,Y) :- bar(X), max(Y).
```

More PROLOG notation: commas (`,`) outside lists denote conjunction. `:-` (to be read as 'if') denotes implication.

Another eight rules (not shown here) suffice to describe the appropriate choices of specifiers (`specifiers(X1)`) and complements (`complements(X)`) for each category. For completeness, we should mention that an additional twelve to twenty rules (also not shown here) will be required to cover empty categories and adjunction structures.

1.6.2 Principles

A variety of formalisms have been used to encode linguistic principles, including Attribute Grammars (Correa [16]), PASCAL (Wehrli [58]), LISP (Kashket [32] and Dorr [20]), PROLOG (Crocker [17], Johnson [31] and Sharp [48]). As mentioned earlier, Stabler has described a theorem-proving system based on a first order logic formulation, but that system is not a parser. However, if theory maintenance is to become practical, we believe that it will be necessary to choose a representation that is as close as possible to the descriptions found in the linguistics literature. This means that the representation should provide the essentially the same "vocabulary", e.g. primitives and combinatory elements, used by linguists so that the formalization process can become as close to a simple one-to-one mapping as possible. In the current representation, there are several compromises due to implementation considerations (to be described in chapter 3). However, we believe that many of the definitions used in the current implementation are straightforward translations of the original definitions. We will illustrate this with a few examples. Many other examples can be found elsewhere in the main chapters of the thesis. In particular, chapter 3 discusses the representation in considerable detail. Chapter 6 also covers others in the context of principle interleaving. All definitions including the linguistically-motivated primitives have been written in PROLOG, a programming language based on a subset of 1st order logic. Whilst PROLOG has its disadvantages as a grammar representation (which will be discussed in chapter 2), we believe that perspicuous definitions that are readily understandable by linguists can be written.

The Case Filter

We will begin with perhaps the simplest principle implemented in the current system. Here is the statement of the filter given in Lasnik & Uriagereka:

(16) At S-structure, every lexical NP needs Case. *(from ch.1, no.20)*

Case, e.g. nominative, accusative, genitive and oblique, is normally assigned to all *lexical*, i.e. non-empty, NPs. For example, in English objects of verbs seem to take accusative Case, *John saw him/*he.* We will not go into the details here (see chapter 3), but various facts such as the distinction between *it is likely that John is here* and **it is likely John to be here* can be explained using the Case Filter in conjunction with the conditions of Case assignment which permit lexical NPs to receive Case only in certain configurations. Compare (16) with the following formulation:

```
:- caseFilter in_all_configurations CF where
       lexicalNP(CF) then assignedCase(CF).
```

```
lexicalNP(NP) :- cat(NP,np), \+ ec(NP).
assignedCase(X) :- X has_feature case(Case), assigned(Case).
```

Some PROLOG notation: (`\+`) denotes negation as failure (to prove).

The first statement represents a universally quantified formula over constituent structure. It is meant to be read as follows: "in every configuration `CF` where `lexicalNP(CF)` holds, then `assignedCase(CF)` must also hold." The next two clauses define the meaning of the terms *lexical NP* and (to be) *assigned Case*:

1. The first clause states that:

   > "the category `NP` is a lexical NP if it has the category label `np`, and it is not an empty category (`ec/1`)."

   Both `cat/2` and `ec/1` are primitives in the system.

2. The next clause also makes use of another two primitives `has_feature/2` (which tests whether a constituent has the feature in question) and `assigned/1` (which tests if some feature slot has been bound). Hence, `assignCase/1` can be read as:

   > "`X` is assigned Case if `X` has a Case feature with slot `Case` and that slot has been filled."

Note that in standard PROLOG, conjunctive goals are executed from left to right. This is unimportant for `lexicalNP/1`. However, `assignedCase/1` relies on the Case slot being identified before it is tested. Finally, we have assumed here that the structures passed to `caseFilter/1` will be from the level of S-structure.

Binding Condition A

We will now sketch how Binding condition A, which constrains the possible coreference options for anaphors, may be perspicuously encoded. Only the top-level portions of the relevant definitions will be supplied here, a more complete definition will be given later in section 3.5.

The following statement of the condition given in Lasnik & Uriagereka is a more sophisticated version of the one introduced at the beginning of the chapter:

(17) An anaphor must be (A-) bound in its governing category (GC).
(from ch.2, no.27a)

The notion of a *governing category* is defined as follows:

(18) The GC of α is the minimal NP or S containing α, a governor of α, and a SUBJECT accessible to α. *(from ch.2, no.105)*

Compare (17) and (18) with the following formulation:

```
:- conditionA in_all_configurations CF
        where anaphor(CF) then gc(CF,GC), aBound(CF,GC).

anaphor(NP) :- cat(NP,np), NP has_feature a(+).

aBound(B,S) :- binds(A,B,S), A has_feature apos.

gcDomain(i2). % i2=S
gcDomain(np).

:- gc(X) smallest_configuration CF st cat(CF,C), gcDomain(C)
        with_components
                X,
                G given_by governs(G,X,CF),
                S given_by accSubj(S,X,CF).
```

The definitions given above should be fairly self-explanatory. For brevity, we have omitted the definitions of the linguistic relations `binds/3`, `accSubj/3` and `governs/3`. The only significant additions are the following:

- *Disjunctive definitions.*

  The disjunction in (18), i.e. "...NP or S ...," has been represented using the two clauses of `gcDomain/1`. This introduces a subtle but important discrepancy between the two definitions. The PROLOG definition implies that the category label of a GC domain must be *either* `i2` *or* `np`. On the other hand, the natural language definition has the possibility of being interpreted as an inclusive-or; in particular, allowing the possibility of having both disjuncts simultaneously satisfied. In this case, there is no effective distinction because linguistic categories have unique labels. However, this is a potentially important difference because Horn clause logic (on which PROLOG is based) cannot represent such inclusive-or disjunctions (that can be represented in 1st order logic). We will return to consider whether this is an important limitation for the representation of principles in chapter 2.

- *Minimality.*

  The `smallest_configuration` form implements definition (18). It states that:

  > "The GC of `X` is the smallest configuration `CF` such that `CF` has category label `C` and `C` is a GC domain, and `CF` contains three com-

> ponents: (1) `X` itself, (2) `G` where `G` governs `X` in `CF`, and (3) `S` where `S` is a SUBJECT accessible to `X` in `CF`."

This statement defines the binary relation `gc/2` referenced in the definition of condition A. In general, `gc(CF,GC)` will hold if `GC` is the GC of `CF` as specified above.

Observe that nowhere in the definitions presented have we specified how a principle should be ordered or whether it should be interleaved. Those decisions need not concern the grammar writer since the intended semantics of the definitions are independent from such control considerations.[22] This concludes our overview of the representation used for phrase structure rules and principles. In the next section, we will describe the architecture of the underlying system.

1.6.3 System Organization

The grammar writer need only be concerned with a fraction of the underlying system. For example, the particular phrase structure recovery algorithm used and details of how changes in the control parameters are reflected in the underlying architecture have been (deliberately) kept hidden as much as possible. In this section, we will briefly describe each of these components in turn.

Figure 1.8 shows the major components of the implemented system. The basic inputs to the system as provided by the grammar writer are shown on the right-hand-side. The "output" of the system is a parser specialized with respect to: (1) the linguistic theory in the form of the principles and S-structure rules, (2) the particular language in the form of the parameters and lexicon (plus part of the S-structure rules), and (3) the parser control specified by the user. Note that the system initially recovers phrase structure at the level of S-structure (rather than at Logical Form or D-structure — the other two syntactic levels described earlier in section 1.4.1) for efficiency reasons to be outlined in chapter 4.[23] One disadvantage to specialization is that portions of a parser may have to be re-built if any of these elements change. In the current system, adding a new entry to the

[22] Actually, not all the principles we have described can be interleaved. In particular, any principle that operates on LFs cannot be interleaved with phrase structure construction because the system (initially) recovers phrase structures at the level of S-structure.

[23] Although the S-structure rules are parameterized with respect to head–complement and specifier–head order, there still remains some work to be done with respect to movement before the phrase structure rules can be truly abstract. For example, as described earlier one of the major differences between Japanese and English is that *wh*-movement is reflected at S-structure in English but not in Japanese. In the current system, the possible consequences in terms of what can appear where at S-structure are still spelled out manually in the rule set. (See chapter 4 for the details.) The solution is, of course, to have the grammar writer only specify the rules of D-structure and add an extra compilation step that automatically computes the possible S-structure rules given an appropriate setting of the whInSyntax parameter.

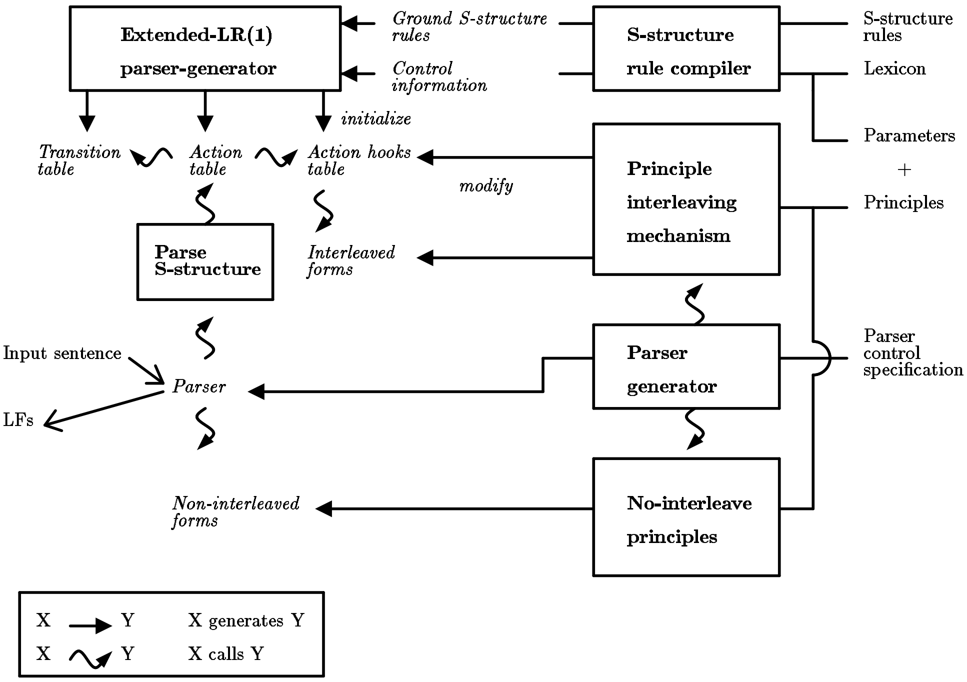


Figure 1.8 The components of the system.

lexicon could require re-computing the entire LR(1)-based automaton. Note that the system makes use of a variety of internal representations (shown in italics) that are normally not visible to the user or grammar writer. We will now briefly explain the significance of each of these:

1. *The phrase structure components.*

   The rules that encode the phrase structure component will be "grounded" with respect to the constituent order parameters [±`specInitial`,±`headInitial`]. Also, $\overline{\mathrm{X}}$ phrase structure rules (introduced previously in section 1.6) will be grounded with respect to particular choices for complements and specifiers for each category. (The compiler will also detect when some of these choices are unavailable for some categories due to the lack of lexical entries of the right type.) The resulting set of ground rules together with extra control information for disambiguation will be input to a canonical LR(1)-based parser generator. The generator produces LR(1)-style transition and action tables. The major differences in the action table are (1) multiple actions for table entries are permitted, and (2) each action may have pre-conditions attached.

These pre-conditions are accessed indirectly via a mapping table *Action hooks table* that can be dynamically modified by other components of the system (as we shall see below).

The phrase structure components have been designed to be modular. It is a simple matter to replace the "back-end" (i.e. the LR(1)-based parser-generator), with an alternate parser-generator using a different algorithm. For example, in an earlier version the intermediate representation, i.e. the ground rules, was not compiled, but simply interpreted.

2. *The principle compilation components.*

   The parser control specifies which principles are to be called independently or interleaved with structure building. The compiler will generate the appropriate forms in either case:

   (a) If a principle is not to be interleaved, a parser operation that accepts a complete phrase structure as input will be generated. This operation will be directly invoked by the top-level parser control (to be described below).

   (b) Otherwise, when a principle is to be interleaved, a predicate that accepts only partial structures as input will be generated. The type inference mechanism will then compute where to "splice in" links to the generated predicate in the LR(1) action table. Note that only one copy of the action table is ever generated. That is, the interleaver does not directly modify action table entries, but instead inserts the relevant calls via the mapping table *Action hooks table*. Note also that this implies that interleaved principles will not be directly called by the parser control. As the dependencies in the system diagram indicate, parser control invokes the operation *Parse S-structure* which will call the appropriate action table entries which, in turn, will invoke the interleaved forms (via the mapping table).[24]

3. *The parser control component.*

   Given a parser control specification, the parser control component generates a top-level control predicate *Parser* that simply calls *Parse S-structure* to obtain phrase structures and passes them to the non-interleaved forms in the order specified.[25] If the non-interleaved forms are to be dynamically ordered, the top-level predicate also calls a secondary control predicate (not shown) for

[24] Note that *Parse S-structure* must also accesses the lexicon (the link is not shown in the system diagram).

[25] The actual implementation is a bit more complicated. Each principle is actually called indirectly via a mapping table similar to that for the LR(1) action table. Generally, there will be a

each structure recovered by *Parse S-structure*. This control predicate manages the list of non-interleaved operations and invokes the operations in the order most likely to rule out the given structure.

4. *The parser.*

   The top-level predicate generated by the parser control component is a two-place predicate that accepts a sentence from the user and returns zero or more Logical Forms.

This completes our review of the components of the system. In the next and final section of the chapter, we will briefly outline the topics covered in each of the remaining chapters.

1.7 Roadmap of the Thesis

The main body of the thesis consists of the following six chapters:

- *Parser Design.*

  Chapter 2 examines the important design decisions and points out where necessary tradeoffs have been made in the current implementation. For instance, it considers the consequences of adopting a simple representation of linguistic principles built on top of PROLOG. The chapter will also make use of comparisons with other principle-based systems to highlight particular design points.

- *The Representation of Linguistic Principles.*

  Chapter 3 describes how linguistic principles are represented and identifies the limitations of the current representation using case studies involving Case theory, chain formation, free indexation and Binding theory.

- *The Recovery of Phrase Structure.*

  Chapter 4 describes how the system copes with multiple levels of phrase structure, ambiguity, and infinite structures. It also documents how the phrase structure rules are compiled into an efficient canonical LR(1)-based machine by specializing the rules with respect to the language and the lexicon. Also, the suitability of the canonical LR(1)-based procedure and the effect of varying language parameters will be examined. Comparisons will also be made to

variety of preconditions and postconditions defined on each principle including the updating of various counters of the type described earlier in the chapter, calls to display the structure passed to the principle — for debugging purposes, extra control that the user can specify, e.g. to just return the first parse instead of looking for all solutions, to return only (or no) structures satisfying some form, and so on.

the LALR(1) procedure, a model that trades the amount of disambiguation possible from lookahead for reduced table size.

- *Principle Ordering.*

  Chapter 5 considers the question of ordering principles for efficient parsing. More precisely, the following questions will be addressed: (1) Will re-ordering principles make a big difference in parsing speed? That is, is principle ordering something parser designers should even be worried about? (2) If so, which orderings are better than others? Can we explain why? (3) Finally, is there an "optimal" ordering of principles that parsers should adopt? Empirical data will be presented to validate the adopted computational model of principles as simple filters and generators. The chapter will also describe and evaluate the use of an ordering mechanism that attempts to dynamically specialize the filters-and-generators model with respect to particular inputs.

- *Principle Interleaving.*

  Chapter 6 will describe the type inference procedure and other components that make up the off-line principle interleaving mechanism. The notion of the size of the type of a principle will be introduced as a basis for predicting the effectiveness of interleaving on particular principles. Empirical data will be presented to correlate the type size with parsing efficiency and to determine which principles are appropriate candidates for interleaving.

- *Conclusions.*

  Chapter 7 discusses the lessons learned from building the system, things that should have been done differently, and points out areas where future work could be profitably applied.

Finally, we will briefly outline the contents of the two appendices also included in the thesis:

- *The Linguistic Theory.*

  Appendix A contains a compact, but complete description of the linguistic theory that was formalized. Since we do not repeat the linguistic arguments for each principle listed, it is not meant to be a substitute for Lasnik & Uriagereka. We have attempted, however, to provide a reference (where possible) to the original source for each definition.

- *The Notation for Encoding Principles.*

  Appendix B contains a complete description of the "programming language" (i.e. the linguistically-motivated primitives and combinative elements), that were used to encode the theory described in appendix A. Type equations or type inference rules used by the principle interleaving mechanism will also accompany all appropriate entries.

Chapter 2

Parser Design

The main chapters of the thesis will be devoted to describing the specific features and components of the implemented system. The purpose of this chapter will be to discuss general design issues for principle-based parsers and to highlight the differences between this system and the approaches taken in other principle-based systems. The remainder of the chapter will be organized around the following topics:

- *Modifying linguistic theory.*

  We will discuss various reasons why grammar writers might want to modify linguistic definitions. We will also outline the position taken in the current system as to the circumstances under which modifications can be made.

- *The level of representation of linguistic principles.*

  We will discuss the consequences of adopting PROLOG as the underlying language for the representation of principles. In general, there seems to be a tradeoff between the abstraction level of the representation and the efficiency of the resulting parser. In the context of this tradeoff, we will consider the question of whether a more powerful formalism such as 1st order logic, as adopted by Stabler [52], is really necessary in order to parse with principle-based theories.

- *Free overgeneration, separation of principles, and efficiency.*

  Next, we will discuss the reasons for, and the problems of, adopting a policy of maintaining a strict separation between principle definitions in the present system.

- *Derived principles.*

  Correa's attribute grammar-based formulation ([16]) is an example of a parser that has not imposed a strict separation between principles. Because Correa's parser and the system described here are based on a common theory of functional determination, we shall directly compare our representation to Correa's reformulated version.

- *Licensing parsers.*

  We will briefly discuss the concept of a *licensing parser*, i.e. a parser that only build structures in accordance with a distinguished subset of principles, as exemplified by Abney [2] (also Frank [24]). We will relate part of Abney's system to ours by illustrating how the control of the current system can be set up in a similar fashion so that the phrase structure recovery component only recovers structures satisfying a designated subset of principles.

- *Methods for efficient parsing.*

  Next, we will briefly survey some of standard compilation and interpretation methods that have been used for parsing. In each case, we will point out the specific advantages and disadvantages of the method for principle-based parsing, and whether the method is utilized in the current implementation.

- *Other issues.*

  Finally, we will finish the chapter with by briefly discussing the role of the notions *directness* and *faithfulness* in the current implementation.

2.1 Modifying Linguistic Theory

- *Linguistic theory is always changing.*

  In recent years the principle-and-parameters framework has proved to be a very productive area of research in linguistics. It is probably safe to assume that linguistic theory will continue to be updated at a fairly rapid rate in years to come. This state of affairs has obvious important implications for parser design. Given the inevitable lag in time that occurs between publication of a theory and its possible realization, most implementations are probably obsolete before they are completed, or perhaps even begun. As we will see below, this will lead us to reject certain techniques such as hand-optimization of principles in favour of less efficient, but more easily maintainable representations.

- *Linguistic theory should not be changed.*

  It can be tempting for the grammar writer to alter principle definitions for a number of reasons:

  1. Principles are usually not written in the most efficient form for computation. In some cases such as *free indexation* (the assignment of indices

to establish coreference relations), as will be discussed in chapter 3, definitions may not be directly *constructive*. That is, principles may not be stated in a form that immediately suggests a corresponding computational procedure.

2. Some linguistic theories may make use of powerful operations on phrase structure that may lead to parser non-termination. A good example of this is the γ-marking proposal of Lasnik & Saito [34]. This scheme makes use of an operation that allows certain elements in phrase structure to move to another position (and possibly back again) without generating an trace. Thus, it becomes possible to derive certain phrase structures in an infinite number of ways.

3. Restatement may also be forced by peculiar limitations in the underlying architecture of the parser. For example, the current system makes extensive use of enumerative procedures that generate all possible hypotheses to be filtered out at a later stage. A problem occurs if an enumeration procedure defines an infinite number of hypotheses. Although, it may be the case that there are only a finite number of valid hypotheses, i.e. all but a finite number of hypotheses will be filtered out at some later stage, the simple generate-and-test strategy will not terminate. For example, the Verb Phrase (VP) adjunction operation (introduced in the previous chapter) may be used to implement the 'scrambling effects' present in languages such as Japanese. Unrestricted application of VP-adjunction will result in an infinite number of (distinct) phrase structures, e.g. $[_{VP}\ [_{NP}\ nani\text{-}o\ ]_i\ [_{VP}\ t_i\ \ [_{VP}\ t_i\ \ldots\ [_{VP}\ t_i\ katta\ ]]]]$ can be generated from $[_{VP}\ [_{NP}\ nani\text{-}o\ ]\ katta\ ]$.

Hence, an important question to ask is: Under what circumstances should it be acceptable to recast definitions to better suit parsing needs?

This thesis takes the position that the grammar writer should avoid changing the definition of a principle unless it is crucial for ensuring parser termination, as in the case of VP-adjunction and γ-marking. In particular, we will argue below that rewriting an already terminating operation simply for efficiency should not be done manually, but only if there exists some well-defined automatic and transparent procedure that accomplishes the task. We are assuming, of course, that systems constructed in this fashion should be more robust. Also, optimizations for efficiency are less likely to prove to be idiosyncratic (and transient) in nature when the principles are modified.

We should mention that one limitation that results from forcing parser operations to be finitely enumerable is that correctness of the restricted operation is not guaranteed. For example, the current system allows only one iteration of repeated

VP-adjunction.[1] So [*VP* [*NP nani-o*]$_i$ [*VP* t_i *katta*]] would be allowed, but not [*VP* [*NP nani-o*]$_i$ [*VP* t_i [*VP* t_i *katta*]]]. Hence, the parser will (incorrectly) miss a parse if there happens to be one that involves two consecutive traces of this type. Basically, what is needed is some sort of *inductive proof capability* in the sense of showing that an ill-formed parse cannot be 'repaired' by (repeated) VP-adjunction.[2] The current system lacks this capability.[3]

2.2 The Level of Representation of Principles

We have found PROLOG ([51]) to be a reasonable compromise between a completely declarative representation such as 1st order logic (FOL) (for which practical general search strategies do not exist) and a more efficiently executable but less abstract model such as that of PASCAL or LISP. By building a suitable collection of predicates that operate on constituent structure, as chapter 3 will show, it is possible to produce executable specifications with little effort for many principles. However, PROLOG does have several flaws that make themselves felt in the principle definitions we have used, thereby preventing it from being an ideal representation language. We will briefly list the major ones:

1. PROLOG is based on Horn clause logic (a limited subset of FOL). In particular, Horn clauses are restricted to 1st order formulas of the form $\forall x_1, \ldots, \forall x_n (L_1 \wedge \neg L_2 \wedge \neg L_m)$ where $x_1, \ldots, x_n$ are the variables occurring in the formula, and $L_1, L_2 \ldots, L_m$, $m \geq 1$, are all positive literals, i.e. no negated atoms. (PROLOG shorthand for the above formula is $L_1 : -L_2, \ldots, L_m$.) Because of the stipulation that all Horn clauses must have at exactly one positive literal, Horn clauses cannot be used to express negative information, e.g. $\neg whInSyntax$ would not be a Horn clause. However, the same stipulation allows the use of the efficient SLD-resolution (Linear resolution with Selection function) strategy (which has been shown to be both sound and complete).

[1] The LR(1)-based parser-generator automatically deals with simple cases of infinite loops. The current system detects and repairs simple cases in the CFSM (characteristic finite state machine) when it is possible to loop back to the same state without consuming any input, as will sometimes occur with λ-rules such as NP$\rightarrow\lambda$ or Adv$\rightarrow\lambda$. (See the screen snapshot in chapter 7 for an example.) The default 'repair' is to only allow it to loop once. This parameter can be altered on a global or state by state basis to allow an arbitrary upper limit.

[2] Also see section 12 in Stabler [52].

[3] A speculative comment is in order here. Note that a system that implements a constraint propagation scheme may be able to avoid having to impose the finite enumerability restriction, and therefore also avoid having to implement the inductive proof tactic. More concretely, a forward pruning strategy (see van Hentenryck [28]) may be able to eliminate the infinite branches in the search space corresponding to, say, repeated VP-adjunction. We have not investigated the application of this technique in principle-based parsing.