ON MEMORY LIMITATIONS IN NATURAL LANGUAGE PROCESSING

by

Kenneth Ward Church

June 1980

This research was supported (in part) by the National Institutes of Health Grant No. 1 P41 RR 01096-02 from the Division of Research Resources and by the National Institutes of Health Grant No. 1 P01 LM 03374-01 from the National Library of Medicine.

ON MEMORY LIMITATIONS IN NATURAL LANGUAGE PROCESSING

by

Kenneth Ward Church

submitted in partial fulfillment
of the requirements
for the degree of
Master of Science
in Electrical Engineering and Computer Science

ABSTRACT

This paper proposes a welcome hypothesis: a computationally simple device is sufficient for processing natural language. Traditionally it has been argued that processing natural language syntax requires very powerful machinery. Many engineers have come to this rather grim conclusion; almost all working parsers are actually Turing Machines (TM). For example, Woods specifically designed his Augmented Transition Networks (ATNs) to be Turing Equivalent. If the problem is really as hard as it appears, then the only solution is to grin and bear it. Our own position is that parsing acceptable sentences is simpler because there are constraints on human performance that drastically reduce the computational requirements (time and space bounds). Although ideal linguistic competence is very complex, this observation may not apply directly to a real processing problem such as parsing. By including performance factors, it is possible to simplify the computation. We will propose two performance limitations, *bounded memory* and *deterministic control*, which have been incorporated in a new parser YAP.

Thesis Supervisor: Dr. Peter Szolovits

Title: Assistant Professor of Electrical Engineering and Computer Science

Keywords: Parsing, Natural Language, Psychological Reality, Finite State, Determinism, Memory Limitations, Performance, Processing

Acknowledgments

- Peter Szolovits, my advisor, for encouraging me to study a new field and seeing me through it even when we both felt out of place

- Mitch Marcus for praising my most outlandish ideas (three times each) even when they conflicted with his own way of thinking

- Bill Martin for believing the problem isn't half as hard as one might think he thinks it is

- Bob Berwick for genuine encouragement and fruitful debate

- Dave McDonald for getting me started (even if we didn't know it at the time)

- Jon Allen, Joan Bresnan, Jay Keyser, Ed Walker and the Cognitive Center for successfully bringing us all together. It is important to have place to formulate ideas and gain appreciation for alternative approaches. I want to especially thank Jon Allen for devoting his seminar on natural language to this purpose.

- Joan Bresnan, Ken Hale, Noam Chomsky, Jay Keyser, and the first year linguistics class for teaching me syntax. I would like to especially thank Joan for her time and her patience.

- Ronald Kaplan (whom I have never met) and Per-Kristian Halvorsen for some brief but instructive conversations via the ARPA network

- Kenneth Wexler, Howard Lasnik and Bob Berwick for some enlightening remarks regarding the goals of linguistic inquiry

- Vaughan Pratt for an appreciation of computational complexity

- my fellow students and co-workers: Ramesh Patil, Bill Swartout, Dave Shipman, Brian Smith, Glenn Burke, Lowell Hawkinson, Harold Goldberger, Howard Sherman, and Greg Faust for the kind of advice that you can't get in class. I want to especially thank Ramesh, Dave and Bill for reading countless drafts.

- the first year linguistics students for their warm friendship

CONTENTS

FIGURES

1. Introduction

This paper proposes a welcome hypothesis: a computationally simple device[1] is sufficient for processing natural language. Traditionally it has been argued that processing natural language syntax requires very powerful machinery. Many engineers have come to this rather grim conclusion; almost all working parsers are actually Turing Machines (TM).[2] For example, Woods specifically designed his Augmented Transition Networks (ATNs) to be Turing Equivalent.

(1) "It is well known (cf. [Chomsky64]) that the strict context-free grammar model is not an adequate mechanism for characterizing the subtleties of natural languages ... When conditions and actions are added to the arcs, the model attains the power of a Turing machine, although the basic operations which it performs are 'natural' ones for language analysis. Using these conditions and actions, the model is capable of performing the equivalent of transformational analysis without the need for a separate transformational component." [Woods70][3]

If the problem is really as hard as it appears, then the only solution is to grin and bear it. Our own position is that parsing acceptable sentences is simpler because there are constraints on human performance that exclude all the "harder" cases. A <u>real</u> parser can take advantage of these performance constraints (e.g. limited memory) so that it can be simpler and more efficient than Woods' <u>ideal</u> model which is designed to parse the entire competence grammar.

1. Throughout this work, the <u>complexity</u> notion will be used in its computational sense as a measure of time and space resources required by an optimal processor. The term will not be used in the linguistic sense (i.e. the size of the grammar itself). In general, one can trade one off for the other, which leads to considerable confusion. The size of a program (linguistic complexity) is typically inversely related to the power of the interpreter (computational complexity). This point is discussed more thoroughly in chapter 6.

2. It is important to distinguish *computational complexity* (time and space bounds) from *computational class* (finite state FS, context free CF, context sensitive CS, turing machine TM). A grammar that describes a large class is generally more difficult to process than a more tightly constrained grammar. For example, FS grammars can be parsed with bounded space; all others consume unbounded space. Similar comments probably hold for time complexity, too (though the proof is an open problem.) That is, FS grammars can be parsed in linear time, whereas CF grammars probably require more time in the worst case.

3. In fairness to the ATN and Transformational [illegible]mar, it should be noted that there have been efforts to reduce the generative capacity. For example, Kaplan (personal communication), [Woods73] and [Peters and Ritchie73] discuss various restrictions to assure decidability. Unfortunately, this move is not sufficient to guarantee efficient (e.g. polynomial time) processing; parsing decidable grammars may be *effective*, but it is hardly *efficient*.

1.1 The Competence/Performance Dichotomy

The approach crucially depends on performance constraints to shrink the search space of possible derivations. Formerly engineers such as Woods attempted to model competence without performance constraints, and not surprisingly, they found they needed inordinate resources to do so. We suggest that a real processor incorporates both competence (grammar) and performance (time and space) constraints. Hence it is possible to build a small efficient processor by exploiting the performance model. This is particularly clear from Chomsky's original description of the performance/competence dichotomy.

(2) "Linguistic theory is concerned primarily with an ideal speaker-listener, in a completely homogeneous speech-community, who knows its language perfectly and is unaffected by such grammatically irrelevant conditions as *memory limitations*, distractions, shifts of attention and interest, and errors (random or characteristic) in applying his knowledge of the language in actually performance .... We thus make a fundamental distinction between *competence* (the speaker-hearer's knowledge of his language) and *performance* (the actual use of language in concrete situations)." [Chomsky65, pp 3-4, italics added]

The proposed model is more efficient and more restrictive than Woods' ATN. It is more efficient because it doesn't have the resources to waste and it is more restrictive because it doesn't have the resources to explore as many possibilities. For example, there are some sentences which will require a very long time on an ATN; our model will reject these sentences as *unacceptable* (not in the performance model) because it doesn't have the time to figure it out. We believe there are two reasons for rejecting sentences; a sentence may be *ungrammatical* (excluded from competence) or it may be *unacceptable* (excluded from performance).[4] The term *acceptable* was coined by Chomsky to refer to:

(3) "... utterances that are perfectly natural and immediately comprehensible without paper-and-pencil analysis, and in no way bizarre or outlandish ... The more acceptable sentences are those that are more likely to be produced, more easily understood, less clumsy, and in some sense more natural. The unacceptable sentences one would tend to avoid and replace by more acceptable variants, wherever possible in actual discourse." [Chomsky65, pp. 10].

4. This position should be distinguished from Kaplan's hypothesis (personal communication) that the processing grammar is identical to the competence grammar. We suggest that there are some extra-grammatical factors (e.g. memory limitations) which distinguish the two.

Acceptability is assigned independently from grammaticality; the four logical possibilities are illustrated by (4).[5]

(4) It is raining.
#Tom figured that that Susan wanted to take the cat out bothered Betsy out.
*They am running.
*#Tom and slept the dog.

Chomsky formulated this distinction in order to separate irrelevant processing constraints (e.g. limited time and space) from the grammaticality questions which he has been studying. Our hypothesis that a simple device can process language, is then, by definition, a hypothesis about the performance model. Acceptability judgments will bear crucially on the matter.[6]

The problem is to design a parser that *approximates* competence with realistic resources. Unacceptable sentences should be excluded because they require inordinate resources to process; ungrammatical sentences should be rejected because they violate competence idealizations (or approximations thereof). The design criteria are summarized below:

(5) What are some reasonable performance approximations?
(6) How can they be implemented without sacrificing linguistic generalizations?

1.2 The FS Hypothesis

We will assume a <u>severe</u> processing limitation on available short term memory (STM), as commonly suggested in the psycholinguistic literature ([Frazier79], [Frazier and Fodor78], [Cowper76], [Bresnan78], [Kimball73, 75], [Chomsky61]). Technically a machine with limited memory is a finite state machine (FSM) which has very nice computational properties when compared to an arbitrary TM. Most importantly, a FSM requires less time and space in the worst case. There are some other advantages which we have not explored

5. These examples are taken from [Kimball73]. A hash mark (#) is used to indicate unacceptability; an asterisk (*) is used in the traditional fashion to denote ungrammaticality.
6. Just as Chomsky idealized grammaticality from other unexplained irrelevant factors, it will be useful to idealize acceptability. In this work, we are most interested in time and space behavior *in the limit* as sentences grow; we will not address borderline cases where judgments tend to be extremely variable. This move is often taken in complexity arguments which study limiting growth, but ignore constants (borderline cases).

in detail. For example, it is easier to run a FSM in reverse. This may have some important implications if one were attempting to build a single model for both production and generation as suggested in [Kay75].[7]

When discussing certain performance issues (e.g. center-embedding),[8] it will be most useful to view the processor as a FSM; on the other hand, competence phenomena (e.g. subjacency)[9] suggest a more abstract point of view. Because of a lack of TM resources, the processor cannot literally apply rules of competence; rather, it resorts to more computationally realistic approximations. Whenever a competence idealization calls for inordinate resources, there will be a discrepancy between the competence idealization and its performance realization.

1.2.1 Center-embedding

Chomsky and Bar-Hillel independently showed that (arbitrarily deep) center-embedded structures require unbounded memory [Chomsky59a,b] [Bar-Hillel61] [Langendoen75]. As predicted, center-embedding is severely compromised in performance; it quickly becomes unacceptable, even at relatively shallow depths.

(7) #[The man [who the boy [who the students recognized] pointed out] is a friend of mine.]

(8) #[The rat [the cat [the dog chased] bit] ate the cheese.]

7. Trivially all physical machines are FSMs. The FS hypothesis is interesting, though, because the memory limitation is so severe (i.e. two or three clauses) that it is a crucial issue in many practical situations. Similar comments can be made about modern computers. Most engineers would model a typical large computer system as a TM. However, it would be hard to think of a computer as a TM if it had only 1 bit of memory. How much memory does it take before a FSM is best modeled as a TM? The answer may depend on the current price of memory. What once seemed unreasonable, may not be so unrealistic today.

8. A center-embedded sentence contains an embedded clause surrounded by lexical material from the higher clause: $[_S\ x\ [_S \ldots]\ y]$, where both *x* and *y* contain lexical material.

9. *Subjacency* is a formal linguistic notion which constrains the applicability of a transformation. (Informally, subjacency is a locality principle; all transformations must be local to a single cyclic node (e.g. clause) or to two adjacent cyclic nodes.) We offer subjacency as an example of a competence idealization. In general, though, it is extremely difficult to prove that a particular phenomenon is necessarily a matter of competence. We have no proof that subjacency is a competence universal, and similarly, we have no proof that center-embedding is a processing universal. Our assessments are most plausible, though conceivably, they might be incorrect.

A memory limitation provides a very attractive account of center-embedding phenomena (in the limit).[10]

(9) "This fact [that deeply center-embedded sentences are unacceptable], and this alone, follows from the assumption of finiteness of memory (which no one, surely, has ever questioned)." [Chomsky61, pp. 127]

1.2.2 Respectively

What other phenomena follow from a memory limitation? Center-embedding is the most striking example, but it is *not* unique. There have been many refutations of FS competence models; each one illustrates the point: *computationally complex structures are unacceptable.* Consider the *respectively* construction[11] which is notorious for its crossing dependencies.[12] As predicted, it too becomes rapidly unacceptable.

(10) John and Jack knew Tim and Mike, respectively.
?John, Jack and Sam knew Tim, Mike and Rob, respectively.
??John, Jack, Sam, and Tom knew Tim, Mike, Rob and Bill, respectively.
???John, Jack, ..., Sam, and Tom knew Tim, Mike, ..., Rob and Bill, respectively.

1.2.3 Lasnik's Noncoreference Rule

Lasnik's noncoreference rule [Lasnik76] is another source of evidence.[13] The rule observes that two noun phrases in a particular structural configuration are noncoreferential.

10. A complexity argument of this sort does not distinguish between a depth of three or a depth of four. It would require considerable psychological experimentation to discover the precise limitations. This account predicts that all center-embedded structures eventually become unacceptable although it is possible that certain constructions become unacceptable more rapidly than others. For example, [Cowper76] has found some differences between relative clauses and complement clauses.

11. [Bar-Hillel61] argued that *respectively* proves the competence model is not CF. It has been widely suggested that *respectively* is really a semantic issue which shouldn't concern syntax. Although this point is well taken, there are numerous analogous constructions (Dutch verbs [Huybregts76], Swedish wh-movement, and Mohawk [Postal64]) which pose the same problem. If all of these arguments are mistaken and grammar is in fact only CF, then it is even easier to defend the FS Hypothesis. (Only center-embedding would have to be excluded.)

12. Crossing dependencies are beyond CF complexity. The proof uses the pumping lemma. [Huybregts76]

13. It can be argued that this rule is not a syntactic rule and hence it is irrelevant to the FS hypothesis. Actually, we believe that the FS hypothesis is more general; it applies at *all* levels of linguistic processing, not just the syntactic component.

(11) The Noncoreference Rule: Given two noun phrases NP_1, NP_2 in a sentence, if NP_1 precedes and commands[14] NP_2 and NP_2 is not a pronoun, then NP_1 and NP_2 are noncoreferential.

For example, each *John* in (12) must refer to different people, since the first *John* both *precedes* and *commands* the second. This rule has unbounded consequences; it applies even when there are an arbitrary number of clauses between NP_1 and NP_2. Consequently, unbounded memory is required to process the rule; it becomes harder and harder to enforce as more and more names are mentioned. His rule is part of a competence model; in performance, it seems necessary to approximate the rule. As the memory requirements grow, the performance model is less and less likely to establish the noncoreferential link. In (12), the co-indexed noun phrases cannot be coreferential. As the depth increases, the noncoreferential judgments become less and less sharp, even though (12)-(14) are all equally ungrammatical.[15]

(12) *#Did you hear that $John_i$ told the teacher that $John_i$ threw the first punch.
(13) *??Did you hear that $John_i$ told the teacher that Bill said that $John_i$ threw the first punch.
(14) *?Did you hear that $John_i$ told the teacher that Bill said that Sam thought $John_i$ threw the first punch.

Ideal rules of competence do not (and should not) specify real processing limitations (e.g. limited memory); these are matters of performance. (12)-(14) do not refute Lasnik's rule in any way; they merely point out that its performance realization has some important empirical differences from Lasnik's idealization.

1.2.4 Convergence

On the other hand, there are idealizations which can be realized in performance *without approximations*. For example, it seems that movement phenomena can cross unbounded distances without degrading acceptability. Compare this with the center-embedding and *respectively* examples previously discussed.[16]

14. Informally, a phrase *precedes* phrases to its right. For example, *x* precedes *y* in: ... x ... y ... A phrases *commands* phrases in subordinate clauses. That is, *x* *commands* each *y* in: $[_S$ ... x ... $[_S$ ... y_1 ... $[_S$ ...y_2 ... See [Lasnik76] for more discussion.
15. Some informants report that they noticed noncoreference, but chose to ignore it in the more complex cases. This seems to conflict with our account that it is too difficult to establish the noncoreference links.
16. We explicitly used the same verbs to illustrate the recursive nature of these constructions. They would be more grammatically acceptable if we used different verbs.

(15) There seems to seem ... to be a problem. *move-np*
(16) What did Bob say that Bill said that ... John liked? *move-wh*

We claim that center-embedding and *respectively* demand unbounded resources whereas movement has a bounded cost (even in the worst case).[17] We will argue in chapters 5, 6 and 8 that a machine can process unbounded movement with very limited resources. Movement phenomena (unlike center-embedding) can be implemented in a performance model *without approximation*. It is a welcome result when performance and competence happen to converge, as in the movement case; there will be no empirical differences between the idealization and its realization. However, there is no logical necessity that performance and competence will ultimately converge in every area. The FS hypothesis, if correct, would necessitate compromising many competence idealizations.[18]

1.3 The Proposed Model: YAP

Some psycholinguists believe there is a natural mapping from ideal competence onto a realistic processing model. This hypothesis is intuitively attractive, even though there is no logical reason that it need be the case.[19] Unfortunately, the psycholinguistic literature does not precisely describe a mapping which is consistent with our FS hypothesis.[20] We have implemented a parser (YAP) which behaves like a complex competence model on acceptable cases, but fails to parse more difficult unacceptable sentences. This performance model looks very similar to the more complex competence machine on acceptable sentences

17. The human processor may not be optimal. The functional argument observes that an optimal processor could process unbounded movement with bounded resources. This should encourage further investigation, but it alone is not sufficient evidence that the human processor has optimal properties.

We claim movement will never consume more than a bounded cost; the cost is independent of the length of the sentence. Some movement sentences may be easier than others. For example, there is considerable experimental evidence suggesting that subject relatives (a) are easier than object relatives (b).

(a) I saw the boy who liked you.
(b) I saw the boy who you liked.

However, we believe the difference between (a) and (b) is independent of their lengths.

18. We have given only three examples: center-embedding, crossing dependencies and noncoreference although there are many more. Center-embedding and crossing dependencies were intended to be illustrative of structural limitations; noncoreference is typical of interpretive rules (such as pronominal binding).

19. Chomsky and Lasnik (personal communication) have each suggested that the competence model might generate a non-computable set. If this were indeed the case, it would seem unlikely that there could be a mapping.

20. Chart parsers (such as GSP [Kaplan73]) do not satisfy our requirements for a psychologically realistic mapping since they are inconsistent with our FS hypothesis. It is not clear how chart parsers can account for the evidence in favor of the FS hypothesis.

even though it "happens" to run in severely limited memory. Since it is a minimal augmentation of existing psychological and linguistic work, it preserves their accomplishments, and in addition, achieves computational advantages. Chapter 2 will discuss the particular augmentation which allows YAP to conserve memory, and hence reduce complexity to that of a FS machine.

The basic design of YAP is similar to Marcus' Parsifal [Marcus79], with an additional limitation on memory. His parser, like most stack machine parsers, will occasionally fill the stack with structures it no longer needs, consuming unbounded memory. To achieve the finite memory limitation, it must be guaranteed that this never happens on acceptable structures. That is, there must be a "forgetting" procedure (like a garbage collector)[21] for cleaning out the stack so that acceptable sentences can be parsed without causing a stack overflow. Everything on the stack should be there for a reason; in Marcus' machine it is possible to have something on the stack which cannot be referenced again. Equipped with its forgetter, YAP runs on a bounded stack even though it is approximating a much more complicated machine (e.g. a PDA).[22]

1.4 Closure Strategies

The forgetting (closure) notion is crucial to this thesis; it enables YAP to parse unbounded structures with only finite memory.[23] There are two closure procedures mentioned in the psycholinguistic literature: Kimball's early closure [Kimball73, 75] and Frazier's late closure [Frazier79] [Frazier and Fodor78]. We will argue that Kimball's procedure is too ruthless, closing phrases too soon, whereas Frazier's procedure is too conservative, wasting memory. Admittedly it is easier to criticize than to offer constructive solutions. Chapter 2 will develop some tests for evaluating solutions, and then propose a compromise which should perform better than either of the two extremes, early closure and late closure, but it will hardly be the final word. The closure puzzle is extremely difficult, but also crucial to understanding the seemingly idiosyncratic parsing behavior that people exhibit.

---

21. The "garbage collection" analogy is not completely accurate. Garbage collectors return storage to the system when it is *known* that it cannot be referenced again; closure procedures return storage when is it *suspected* that it will not be referenced again.

22. A push down automaton (PDA) is a formalization of unbounded stack machines.

23. Bounded memory was the original motivation for closure. Some closure formulations are heuristic; they close a phrase before it is *known* that the phrase in question cannot be referenced again. Theoretically, though, closure need not be heuristic; it is possible for a FSM to parse non-center-embedded CF structures *without heuristics*. We have opted for a heuristic formulation which appears to more practical (as we will argue in the next section).

1.5 Marcus' Determinism Hypothesis

The memory constraint becomes particularly interesting when it is combined with a control constraint such as Marcus' Determinism Hypothesis [Marcus79]. The Determinism Hypothesis claims that once the processor is committed to a particular path, it is extremely difficult to select an alternative. For example, most readers will misinterpret the underlined portions of (17)-(19) and then have considerable difficulty continuing. For this reason, these unacceptable sentences are often called Garden Paths (GP). A memory limitation alone fails to predict the unacceptability of (17)-(19) because GPs don't center-embed very deeply (and hence there exits a FSM which could parse these GP sentences). Determinism offers an additional constraint on memory allocation which provides an account for the data.[24]

(17) # The horse raced past the barn fell.
(18) # John lifted a hundred pound bags.
(19) # I told the boy the dog bit Sue would help him.

There have been many other attempts to capture the same intuitive notion. Kimball's Processing Principle [Kimball73], McDonald's Indelibity Stipulation [McDonald79], and Frazier's "shunting" notion [Frazier and Fodor78] are typical examples from the psycholinguistic literature. The "shunting" notion assigns a high cost to backing up past a phrase that has been "shunted" from one stage to another.

(20) Indelibility: "Once a linguistic decision has been made, it cannot be retracted -- it has been written with 'indelible ink' ... It requires *every* choice made during the production process, at whatever level, cannot be changed once it has been made -- choices must be made correctly the first time." [McDonald79, pp. 16]

(21) "Principle Seven (Processing): When a phrase is closed, it is pushed down into a syntactic (possible semantic) processing stage and cleared from short-term memory." [Kimball73 pp. 39]

24. There are other possible accounts which may be very similar to Marcus' account. For example, GPs are often related to backup in non-deterministic frameworks. However, it is not clear how such an account can distinguish backup on a GP from backup on an acceptable sentence. One solution places a bound on backup to enable the parser to backup on the acceptable sentences but not on GPs. In some sense, this is very similar to Marcus' approach; he provides a bound on lookahead (analogous to bounded backup) which distinguishes GPs from acceptable sentences.

Although the "determinism" notion is widely discussed in the literature, it is extremely difficult to describe precisely. At first we believed the memory constraint alone would subsume Marcus' hypothesis, thus providing a precise independently motivated account. Since all FSMs have a deterministic realization,[25] it was originally supposed that the memory limitation guaranteed that the parser is deterministic (or equivalent to one that is). Although the argument is theoretically sound, it is mistaken.[26] The deterministic realization may have many more states than the corresponding non-deterministic FSM. These extra states are extremely costly and lack empirical justification. They would enable the machine to parse GPs by delaying the critical decision.[27] In spirit, Marcus' Determinism Hypothesis excludes encoding non-determinism by exploding the state space in this way; it assumes that most exploded states are not reachable in performance. This amounts to an exponential reduction in the size of the state space, which is an interesting claim, not subsumed by FS (which only requires the state space to be finite).[28]

The forgetting procedure, which is the subject of chapter 2, will be "deterministic"; it will not backup or undo previous decisions. Consequently, the machine will not only reject deeply center-embedded sentences but it will also reject sentences such as (22) where the heuristic forgetting procedure makes a mistake (takes a garden path).

(22) #Harold heard [that John told the teacher [that Bill said that Sam thought that Mike threw the first punch] yesterday].

Marcus' Determinism Hypothesis predicts that some sentences would be garden paths (since the state space cannot be exploded), but it alone does not identify which sentences are GPs and which ones are not. He proposes a specific parsing model (Parsifal) to identify garden paths. Parsifal makes a single left to right pass over the sentence. It has to decide what to do at each point based upon a limited lookahead of three constituents. According to Marcus, certain sentences require more lookahead to disambiguate algorithmically, and consequently, Parsifal has to guess what to do. In the garden path case, Parsifal guesses incorrectly.

---

25. A non-deterministic FSM with *n* states is equivalent to another deterministic FSM with 2^n states.
26. I am indebted to Ken Wexler for pointing this out.
27. The exploded states encode disjunctive alternatives (as observed in [Swartout78]). Intuitively, GPs suggest that it isn't possible to delay the critical decision; the machine has to decide which way to proceed.
28. Marcus' hypothesis is necessarily vague because there is no clear way to distinguish an exploded state from a primitive state, without reference to a particular machine (grammar). The definition becomes more precise when state assignments are independently motivated (by linguistic generalizations).

The three constituent limit is a very good description; all the garden path sentences shown above would require a four constituent lookahead to disambiguate correctly. (23) illustrates Marcus' account on a typical GP. It would be acceptable if the machine looked ahead another constituent.[29]

(23) # The horse [$_1$ raced] [$_2$ past] [$_3$ the barn] [$_4$ fell].

The three constituent story is not a complete explanation. Why does Parsifal guess that *raced* is a main verb and not a participle? The main verb interpretation is apparently the unmarked (preferred) case. Would it be possible to have a language where the participle reading was the unmarked case?

1.6 Frazier's Principles

Frazier [Frazier79] [Frazier and Fodor78] has attempted to describe the unmarked interpretations. She has proposed two principles which are presumably universal. There is an intuitive functional motivation for these principles; they appear to require fewer resources (memory and backup) than the alternatives. Frazier has provided considerable experimental evidence as empirical verification.

(24) Minimal Attachment: Attach incoming material into the phrase marker being constructed using the fewest nodes consistent with the well-formedness rules of the language.

(25) Late Closure: When possible, attach incoming material into the clause or phrase currently being parsed.

---

29. In practice, the lookahead will consist of noun phrases and single words; the machine does not, for example, build prepositional phrases in the lookahead buffer. Unfortunately this is crucial to Marcus' account of the GP phenomena; Parsifal does *not* analyze sentence (23) as. *The horse [$_1$ raced] [$_2$ past the barn] [$_3$ fell].* If it did, then it would be able to disambiguate the sentence.

There are some other problems with this account. For example, material after the third constituent shouldn't affect the judgments, and yet, the sentences below seem to be more acceptable than (23).

The horse raced past [$_3$ the barn] fell down.
The horse raced past [$_3$ the barn] stumbled.

We have no explanation for this data. Nevertheless, Marcus' account is the best description in the literature; we will accept it for the time being despite its problems.

Frazier's position is basically compatible with Marcus'; her principles define the unmarked actions when there is insufficient lookahead to be certain. Late Closure, which is relevant to the discussion on forgetting, is central to chapter 2; Minimal Attachment is the topic of chapter 4. There are some (rare) cases where the principles fail to find a correct parse on the first pass, forcing backup in her non-deterministic framework. These will be interpreted as marked "counter-examples" in our deterministic FS framework.[30] We will add a few marked rules to cover the exceptions.

1.7 Capturing Generalizations

Having laid out the basic framework (limited memory and determinism), it is worthwhile to gain some breadth. YAP has encoded a competence model strongly resembling the recent work of Bresnan and Kaplan [Bresnan78], [Bresnan80], [Kaplan and Bresnan80]. They use two representations: a constituent structure and a functional structure. The former deals with mother/daughter relationships whereas the latter is concerned with grammatical roles (subject, object, etc.) and syntactic features (case, tense, person, number, gender, etc.) Chapter 3 discusses the YAP implementation of constituent structure, and Chapter 5, the functional structure.

With the Bresnan-Kaplan representation system, it is relatively straightforward to implement many of their analyses. Chapter 6 presents some typical lexical rules (raising and passive), thus capturing many of the generalizations which were once believed to be beyond the capabilities of a FSM.

YAP also shares many properties with Parsifal; it is possible to implement Parsifal-style transformations in YAP. Chapter 7 implements auxiliary inversion and imperative using Parsifal's approach. This demonstrates an alternative method to capture the generalizations that were used to "refute" the FS hypothesis.

There are two classes of transformations which have been traditionally problematic for processing: wh-movement and conjunction. Chapters 8 and 9 present the approach taken in YAP. Conjunction is particularly interesting because it has never been implemented in a Marcus style deterministic parser. Both of these constructions pose many difficult problems; only some of these have been solved. However, YAP has produced some exciting initial results, correctly parsing the following sentences:

30. She is crucially assuming a non-deterministic framework where the processor can backup past certain errors. In our framework, we need some exceptional rules to prevent the processor from taking the wrong path in the first place.

(26) Which boys and girls went?
(27) Which boys and which girls went?
(28) Which boys went to the ball and took the jar?
(29) Which boys went to the ball and to the jar?
(30) What boy did Bill look at and give a ball to?
(31) Bob gave Bill a ball and John a jar.
(32) Bob saw Bill and Sue Mary.
(33) I want Bill, Bob, and John to be nice.

1.8 Limits of This Research

It has not been possible to study all issues relevant to parsing; we have touched on just a few of the many interesting problems. This section will mention some areas for further study.

(34) Coverage
(35) Semantic Interaction
(36) Length Bias (word count)
(37) Lexical Ambiguity

1.8.1 Linguistic Coverage

There are many constructions which will not be discussed; YAP is similar to Marcus' Parsifal in coverage. Both parsers handle a range of fairly difficult phenomena, are intended to handle robustly all interactions among these phenomena, though neither parser has extensive coverage. YAP does not parse (38)-(39), for example.

(38) I am taller than Bill.	*comparative*
(39) The duck is too old to eat.	*tough movement*

We have nothing to say about the internal structure of noun phrases such as (40). It would have been relatively straightforward to replicate Marcus' approach.

(40) a <u>nice</u> man
a <u>fallen</u> leaf
*a <u>given</u> child

a hundred pound bag
# a hundred pound bags

1.8.2 Semantics

YAP does very little semantic processing. For example, YAP does not distinguish between animate and inanimate objects; (41) and (42) are equally parsable from YAP's point of view.

(41) I gave Bill a ball.
(42) I gave a ball Bill.

It is somewhat difficult to distinguish semantics and syntax. YAP does check grammatical relations (subject, object, etc.). (43) and (44) are correctly distinguished because *go* and *see* take different arguments.

(43) I saw Bill.
(44) *I went Bill.

We have not considered bound anaphora and quantifier scope as illustrated below.

(45) Bill saw himself. *bound anaphora*
*Himself was seen.
*Each other were seen.

(46) Bill saw everyone. *quantifier scope*
Everyone was seen by Bill.

1.8.3 Length Bias and Lexical Ambiguity

There are at least two other processing variables that seem to be relevant: length and lexical ambiguity. Both of these are extremely difficult problems which have been widely studied elsewhere [Frazier and Fodor78] [Milne78a,78b,79,80]. (47) provide some evidence that length (i.e. number of words) influences parsing strategies;[31] (48) illustrates some problems with lexical ambiguity.

(47) #The woman the man the girl loved met died. *length*

??The very beautiful young woman the man the girl loved met on a cruise ship in Maine died of cholera in 1962.

Joe brought the book for Susan.

Joe brought the book that I had been trying to obtain for Susan.

(48) They were <u>flying</u> planes. *lexical ambiguity*

The <u>pupils</u> were small.

I love <u>building</u> <u>blocks</u>.

Whatever they are <u>building</u> <u>blocks</u> the view.

All of these issues are extremely important topics for further research.

31. This evidence is from [Frazier and Fodor78]. Much of it is highly controversial; there may be alternative accounts. Nevertheless, even if we can't provide adequate evidence, it is most plausible that length influences parsing strategies.

2. Closure

YAP is essentially a stack machine parser like Marcus' Parsifal with an additional bound on stack depth. This chapter will deal with the stack allocation problem. There will be a forgetting procedure to remove finished phrases from the stack so the space can be recycled. The procedure will have to decide (heuristically) when a phrase is finished (closed).

2.1 On Left/Right Biases

If we are going to count stack depth, we should be very careful that stack depth corresponds to something meaningful. We will assume stack space ought to be correlated with the depth of center-embedding. Empirically, both left and right branching are relatively free in comparison with center-embedding as (49)-(51) illustrate.[32]

32. This position is somewhat different from the "hold hypothesis" [Kaplan75] which accounts for center-embedded relative clauses but no other types of center-embedding. We believe that *all* forms of center-embedding become rapidly unacceptable even at shallow depths. For example, the following sentences from [Rich75] are unacceptable even though there are no center-embedded relative clauses. We accept Rich's argument that the "hold hypothesis" fails to account for *all* of the center-embedding facts.

(a) #I think [claiming [voting Republican] is immoral] is silly.
(b) #I think claiming [the dog [that bit the burglar] is scared] is silly.

Notice that both left and right branching have many "bunched up" brackets. Langendoen (personal communication) has observed that "bunched up" brackets are redundant, and hence they can be deleted without loss of information. In a sense, the FSM manipulates the resulting representation.

Alternatively, one might view the brackets as corresponding to stack instructions. An open bracket ([) is analogous to "push" and close (]) is analogous to "pop". Deleting brackets corresponds to optimizing stack operations (e.g. tail recursion [Steele78]). Just as "bunched up" brackets suggest a redundancy, a sequence of "pop" instructions in the logical flow of a program indicates wasted stack space.

One can view closure as deleting brackets, like tail recursion, to optimize stack usage. In left and right branching, it is possible to delete the "bunched up" brackets, and hence, bound the maximum stack depth. This fails to bound memory requirements in center-embedded cases where there are no "bunched up" brackets to delete. Chomsky's proof [Chomsky59a,b] is a formalization of this intuition; center-embedding cannot be optimized because it requires unbounded memory (there is no way to convert a strictly CF grammar into a FS grammar). On the other hand, it is possible to optimize non-center-embedded structures because they are FS equivalent.

(49) [[[[The man]'s oldest brother]'s best friend]'s sister] ... *left*
(50) #[The man [who the boy [who the students recognized] pointed out] is a friend of mine.] *center*
(51) [The students recognized the boy [who pointed out the man [who is a friend of mine.]]] *right*

Although we consider left and right branching structures to be equally easy to parse, there have been psycholinguistic models with a left/right bias. For example, Yngve [Yngve60] suggested that left branching was more difficult than right branching because left branching is extremely difficult for a left-to-right top-down parser.[33] However, the dual argument could have been used to argue against right branching structures because they would be costly for a left-to-right bottom-up parser. Theoretically, neither left nor right branching requires unbounded memory because Chomsky showed that non-center-embedded CF structures (e.g. left and right branching and combinations thereof) could be processed with a finite state machine [Chomsky59a,b]. On the other hand, center-embedding is provably difficult because it requires unbounded memory; it cannot be processed by a FSM.[34]

It is possible that human processing is not optimal in this way; there might in fact be a left/right bias even though there is no computational motivation. The research strategy taken here will investigate the optimal methods first. Although computationally optimal procedures are not necessarily the ones people do in fact use, they are likely candidates for further research.

One might argue as Yngve has, that English has a left/right bias even though no one has found a computational motivation. In fact, it is very difficult to find acceptable left branching clauses in English. There doesn't seem to be an acceptable left branching paraphrase of (52) in English, as (53) and (54) illustrate. Yngve's left/right processing bias is certainly not universal to all languages because there are languages (e.g. Japanese) where left branching is just as productive as right branching is in English. Hence the left/right asymmetry in English is language specific; it does not indicate a bias in the human processing system.[35]

33. For example, left branching is infinitely difficult (impossible) for an LL(k) parser. It also caused the Harvard Syntactic Analyzer (HPA) [Kuno66] considerable problems.
34. There have been arguments for a left/right asymmetry based on the assumption that human processing is online (left-to-right). Chomsky's 1959 proof shows that these arguments are invalid.
35. We know of no languages which have both left and right branching clauses. This generalization is unexplained if it is indeed universal.

(52) It is interesting that it is indeed true that John likes Mary. *right branching*
(53) #That that John likes Mary is indeed true is interesting.
(54) #John's liking Mary's being indeed true is interesting.

2.1.1 Kuno's Account

Why do clauses tend to branch toward the left in Japanese and toward the right in English? Although there are no known explanations, Kuno [Kuno72] [Kuno74] provides a very attractive functional account of a related phenomenon: [Greenberg63] noticed that VSO[36] languages are prepositional (right branching) and SOV are usually postpositional (left branching). (Kuno's account does not apply in SVO languages like English.)[37]

(55) Universal 3: Languages with dominate VSO order are always prepositional.
(56) Universal 4: With overwhelmingly greater than chance frequency, languages with normal SOV order are postpositional. [Greenberg63]

Kuno observed that Greenberg's principles happen to be optimal; if a language violated Greenberg's principles then it would be more prone to center-embedding and consequently more difficult to process. Consider the case of relative clauses. Kuno observed that relative clauses should precede the head noun phrase in SOV languages and follow the head in VSO language in order to avoid center-embedding. This is very easy to demonstrate. Examples (57) and (58) obey Kuno's observation and avoid center-embedding; all violations do in fact center-embed as (59) and (60) illustrate.[38]

(57) $[S_2\ O_2\ V_2$ that$]\ S_1\ O_1\ V_1$ *not center-embedded*
(58) $V_1\ S_1\ O_1$ [that $V_2\ S_2\ O_2]$

(59) S_1 [that $S_2\ O_2\ V_2]\ O_1\ V_1$ *center-embedded*
(60) $V_1\ S_1\ [V_2\ S_2\ O_2$ that$]\ O_1$

36. S, V and O stand for subject, verb and object. A VSO language has the predominant word order: verb, subject, object.
37. [Frazier80] generalizes Kuno's argument to apply to SOV language, though her assumptions are somewhat more open to dispute.
38. Recall that a center-embedded clause has lexical material on *both* sides of it. In this case, the center-embedded clauses are surrounded by an *S* and an *O*.

Furthermore, notice the complementizer *that* falls between the head noun phrase and the relative clause. This also happens to avoid center-embedding. The alternative would bracket the relative clause between the head noun phrase and the complementizer, forcing center-embedding. Hence, complementizers will precede relative clauses in VSO languages (universal 3) and follow relative clauses in SOV languages (universal 4). By avoiding center-embedding in this way, we have converged on some of Greenberg's principles. [Kuno74] shows how this reasoning can be applied to some other constructions.

This does not explain why languages are this way, but it is an attractive account which should motivate further research to verify Greenberg's empirical observations. Furthermore, Kuno's argument has no left/right asymmetry; only center-embedding is considered costly. It seems that center-embedding is universally difficult whereas left/right biases are language specific consequences of the center-embedding universal.

2.2 Closure Specifications

We will assume the stack depth should be correlated with the depth of center-embedding. It is up to the forgetting procedure to close phrases and remove them from the stack, so only center-embedded phrases will be left on the stack. The procedure could err in either of two directions; it could be overly ruthless, cleaning out a node (phrase) which will later turn out to be useful, or it could be overly conservative, allowing its limited memory to be congested with unnecessary information. In either case, the parser will run into trouble, finding the sentence unacceptable. We have defined the two types of errors below.[39] We will argue that Kimball's Early Closure is premature and Frazier's Late Closure is ineffective.

(61) <u>Premature Closure</u>: The forgetting procedure prematurely removes phrases that turn out to be necessary.

(62) <u>Ineffective Closure</u>: The forgetting procedure does not remove enough phrases, eventually overflowing the limited memory.

39. These definitions happen to have a functional flavor. We use the functional notion "machine" interchangeably with the algebraic notion "grammar". Our definitions should not be taken literally; we do not mean to imply that there is a forgetting procedure in the brain just because it might be convenient. We are merely suggesting that a forgetting procedure is a useful metaphor for modeling the computation that takes place.

2.3 Kimball's Early Closure

The bracketed interpretations of (63)-(65) are unacceptable even though they are grammatical. Presumably, the root matrix[40] was "closed off" before the final phrase, so that the alternative attachment was never considered. Kimball is crucially assuming that closure is possible before the daughters themselves have been completely parsed. Imagine that a node corresponds to a collection of pointers to its daughters; it is finished when all of the pointers are connected. This does not require that the daughters themselves be finished. For example, the node [$_S$ Joe figured [?]] is finished when a pointer is established to the node [?] even though the contents of [?] remain to be discovered.

(63) # Joe figured [that Susan wanted to take the train to New York] out.
(64) # I met [the boy whom Sam took to the park]'s friend.
(65) # The girl$_i$ applied for the jobs [that was attractive]$_i$.

Closure blocks high attachments in sentences like (63)-(65) by removing the root node from the stack long before the last phrase is parsed. For example, it would close the root clause just before *that* in (67) and *who* in (68) because the nodes [$_{comp}$ that] and [$_{comp}$ who] are not immediate constituents of the root. The root clauses would be frozen in the following configurations: [Tom said S-][41] in (67) and [Joe looked NP] in (68). Having closed the root, it shouldn't be possible to reference it again. In particular, nothing else can attach directly to the root.[42] This model inherently assumes that memory is costly and presumably fairly limited. Otherwise, there wouldn't be a motivation for closing off phrases.

(66) Kimball's Early Closure: A phrase is closed as soon as possible, i.e., unless the next node parsed is an immediate constituent of that phrase. [Kimball73]

(67) [$_S$ Tom said
  [$_{S-}$ that Bill had taken the cleaning out ... yesterday
(68) [$_S$ Joe looked the friend

40. A matrix is roughly equivalent to a phrase or a clause. A matrix is a frame with slots for a mother and several daughters. The root matrix is the highest clause.
41. We use an x-bar [Jackendoff77] notation. *s-* (*s* bar) dominates *s* in embedded clauses (*s- -> comp s*). It is also important to notice that the *s-* in [Tom said S-] is not completely finished; it is possible to attach material to the embedded *s-* but not to the closed root.
42. Kimball's closure is premature in these examples since it is possible to interpret *yesterday* attaching high as in: *Tom said [that Bill had taken the cleaning out] yesterday.*

$[_{s\text{-}}$ who had smashed his new car ... up

There is a slight problem with Kimball's formulation which will become important when we propose our own proposal. The *unless* clause should have a second condition to block closure until a phrase has all of its obligatory daughters. For example, taking Kimball's definition literally, s_1 ($[_s$ The boy s_2 ...]) should close before *who* in (69) because *who* is not an immediate constituent of s_1. This would be a mistake because s_1 does not yet have a verb phrase. Closure should wait for all the obligatory daughters. For example, an *s* has two obligatory daughters: a noun phrase and a verb phrase. Consequently, s_1 cannot close before *who* because it doesn't have its obligatory verb phrase.[43]

(69) $[_1$ The boy $[_2$ who the teacher always liked best] did something really awful.]

2.3.1 A Counter-Example

Although Kimball's motivation to save stack space is well-founded, the precise formulation makes an incorrect prediction.[44] If the upper matrix is <u>really</u> closed off, then it shouldn't be possible to attach <u>anything</u> to it. Yet (70)-(71) form a minimal pair where the final constituent attaches low in one case, as Kimball would predict, but high in the other, thus providing a counter-example to Kimball's story. Evidently, the root was closed prematurely in (71) because it is possible to attach *a rotten driver* to it.

(70) I called [the guy who smashed my brand new car up]. *low attachment*
(71) I called [the guy who smashed my brand new car] a rotten driver. *high attachment*

43. A *s* can take a number of optional adjuncts and conjuncts.

44. We have a methodological suspicion about any theory which predicts an unexpected asymmetry. Kimball's principles (as stated in [Kimball73]) have two such asymmetries: his model is both top-down and right associative. It happens that his predictions are basically correct for a right branching language like English, but not for a left branching language such as Japanese [Cowper76, pp. 29-31]. Kimball's principles conflate several phenomena, involving both closure and language type. It ought to be possible to describe the closure phenomenon independently of word order. An ideal explanation would not distinguish between left and right, because some languages are left branching and some are right branching. This is really a rather minor point though; restating the facts in this way should pose no particular problems.

Kimball would probably not interpret his closure strategy as literally as we have. Unfortunately computer models are brutally literal. Although there is considerable content to Kimball's proposal (closing before memory overflows), the precise formulation has some flaws. We will reformulate the basic notion along with some ideas proposed by Frazier.

2.4 Frazier's Late Closure

Suppose that the upper matrix is not closed off, as Kimball suggested, but rather, temporarily out of view. Imagine that only the lowest matrix is available at any given moment, and that the higher matrices are stacked up. The decision then becomes whether to <u>attach</u> to the current matrix or to <u>close</u> it off, making the next higher matrix available. The strategy attaches as low as possible; it will attach high if all the lower attachments are impossible. Kimball's strategy, on the other hand, prevents higher attachments by closing off the higher matrices as soon as possible. In (70), according to Frazier's late closure, *up* <u>can</u> attach[45] to the lower matrix, so it does; whereas in (71), *a rotten driver* <u>cannot</u> attach low, so the lower matrix is closed off, allowing the next higher attachment. Frazier calls this strategy <u>late closure</u> because lower nodes (matrices) are closed as late as possible, after all the lower attachments have been tried. She contrasts her approach with Kimball's <u>early closure</u>, where the higher matrices are closed very early, before the lower matrices are done.

(72) <u>Frazier's Late Closure</u>: When possible, attach incoming material into the clause or phrase currently being parsed.

2.4.1 A Problem: Right Branching

Late Closure is an improvement because it does not close prematurely like Early Closure. Unfortunately, it is too conservative, allowing nodes to remain open (not closed) too long, congesting valuable stack space. Our compromise will modify Frazier's strategy enabling higher clauses to be closed earlier under certain marked conditions. As late closure is defined by Frazier, right branching structures such as (73) and (74) are a real problem.

45. Deciding whether a node <u>can</u> or <u>cannot</u> attach is a difficult question which must be addressed. YAP uses the functional structure [Kaplan and Bresnan80] and the phrase structure rules. For now we will have to appeal to the reader's intuitions.

(73) This is the dog that chased the cat that ran after the rat that ate the cheese that you left in the trap that Mary bought at the store that ...

(74) I consider every candidate likely to be considered capable of being considered somewhat less than honest toward the people who ...

The problem is that the machine will eventually fill up with unfinished matrices, unable to close anything because it hasn't reached the bottom right-most clause. Hence it will find these right branching sentences just as unacceptable as center-embedding. Perhaps Kimball's suggestion is premature, but Frazier's is ineffective. The compromise solution will strongly resemble Frazier's late closure strategy except there will be one marked case of early closure to handle right branching structures.

Our argument is like all complexity arguments; it considers the limiting behavior as the number of clauses increase. Certainly there are numerous other factors which decide borderline cases (3-deep center-embedded clauses for example). We have specifically avoided borderline cases because judgments are so difficult and variable; the limiting behavior is much sharper. In these limiting cases, though, there can be no doubt that memory limitations are relevant to parsing strategies. In particular, alternatives cannot explain why there are no acceptable sentences with 20-deep center embedded clauses. The only reason is that memory is limited; see [Chomsky59a,b], [Bar-Hillel61] and [Langendoen75] for the mathematical arguments.

2.4.2 Analogies from LL(k) and LR(k) Algorithms

It would help to abstract the closure problem in terms of formal parsing algorithms. Among deterministic parsing algorithms, LL(k) parsing corresponds to the <u>earliest</u> possible closing whereas LR(k) corresponds to closing at the latest possible moment.[46] In LL(k), the machine decides to close before any of the daughters have been attached, whereas an LR(k) parser decides to close after all of the daughters have been attached. Kimball's scheme is not quite as ruthless as LL(k); his parser closes after all but the last daughter has been attached. Frazier's scheme is remarkably similar to LR(k), where the closure decisions are made at the last possible moment. Early closing schemes tend to be premature; they cannot parse as many constructions as later closing schemes. In particular, LL(k) cannot parse left recursive expressions. Later closing schemes tend to be ineffective, wasting memory. An LR(k) parser will push all the input onto the stack in the worst case (right branching).[47] Closing early reduces the parser's capabilities whereas closing late increases the memory costs.[48]

It might be noted here that Marcus' parser actually behaves very much like an LR(k) parser in this respect,[49] and hence, like Frazier's scheme.[50] That is, it pushes the entire right-most branch (from the root to the most recently read word) onto the stack, so that it never prematurely closes a node as an LL(k) parser does; on the other hand, it will often waste stack space like an LR(k) parser.

46. Recall that both LL(k) and LR(k) parse CF grammars on a deterministic stack machine (DPDA). LL(k) is purely top-down; the machine decides which production to expand (push onto the stack) given the mother and the next *k* input symbols. The stack is popped when the next input symbol matches the top of the stack. This discovers the left-most derivation (for ambiguous grammars). LR(k) is the dual; the machine decides which production to reduce (pop off the stack) given the next *k* input symbols and the previous state. Input symbols are pushed onto the stack when there are no productions to reduce. LR(k) finds the right-most derivation. LL(k) is a *predictive* parser because it predicts expansions top-down; LR(k) is a *shift-reduce* parser because it decides whether to shift (push an input symbol) or to reduce (pop a production off the stack).

LL(k) are optimal for purely right branching structures; the stack grows infinitely on left branching structures (doesn't halt), and linearly with the depth for center embedding, but is bounded on right branching. LR(k) parsing is the dual; it is optimal for purely left branching structures where the stack depth is bounded. On center and right branching, the stack depth grows linearly. LL(k) is not as general as LR(k), but it is more space efficient when it works. In our terms, LL(k) parsers suffer from premature closure whereas LR(k) parsers may require more memory (ineffective closure).

47. If memory is cheap then LR(k) is very attractive. Currently several computer programming languages are parsed with LR(k) techniques because the memory demands are tolerable. We are assuming that human short term memory limitations are far too severe for such extravagances.

48. [Kimball75] makes a similar point. He offers two compromise points between LL(k) and LR(k) and shows that the corresponding languages are all properly nested. (Both compromises appear to be premature; the arguments are not very interesting.)

49. Marcus himself has argued this point on many occasions (personal communication).

50. Marcus had not been thinking about the closure issue. Nevertheless, his work forms an interesting data point among the possible closure strategies.

2.5 A Compromise

We have designed YAP to close late by default (like LR(k), [Frazier79] and [Marcus79]) with one marked exception to alleviate the memory load (in the right branching case).[51] The marked case of early closure is described by the *A-over-A early closure principle.* It is very much like Kimball's early closure principle except that it waits for two nodes, not just one. For example in (75), our principle would close $[_1$ that Bill said $S_2]$ just before the *that* in S_3 whereas Kimball's scheme would close it just before the *that* in S_2.

(75) John said $[_1$ that Bill said $[_2$ that Sam said $[_3$ that Jack ...

(76) <u>The A-over-A early closure principle</u>: Given two phrases in the same category (e.g. noun phrase, verb phrase, clause, etc.), the higher closes when both are eligible for Kimball closure. That is, (1) both nodes are in the same category, (2) the next node parsed is not an immediate constituent of either, and (3) the mother[52] and all obligatory daughters have been attached to both nodes.

This principle, which is more aggressive than Frazier late closure, enables the parser to process unbounded right recursion within a bounded stack by constantly closing off. However, it is not nearly as ruthless as Kimball's early closure, because it waits for two nodes, which may alleviate the problems that Frazier observed with Kimball's strategy.

There are some questions about the borderline cases where judgments are extremely variable. Although the A-over-A early closure principle makes very sharp distinctions, borderline cases are often questionable. See [Cowper76] for an amazing collection of subtle judgments that confound every proposal yet made. However, we think that the A-over-A notion is a step in the right direction; it has the desired limiting behavior,[53] although the borderline cases are not yet understood. Chomsky comes to a similar conclusion:

51. Early closure is very similar to a compiler optimization called <u>tail recursion</u>, which converts right recursive expressions into iterative ones, thus optimizing stack usage. Compilers would perform optimizations only when they can prove that the structure is right recursive; the A-over-A closure principle is somewhat heuristic because the structure may turn out to be center-embedded.

52. A node can't close until it knows who its mother is. This is important because it is possible in YAP to build nodes bottom-up. They might have all their daughters, but not their mother. Secondly, we assume the root doesn't have a mother and hence it cannot close. This will have some important implications as we will see.

53. Notice that an A-over-A-over-A principle would also have the same limiting behavior. In general, if there are *n* categories, an A-over-A principle would limit the stack depth to *2n* (in the right branching case) whereas an A-over-A-over-A principle would limit the depth to *3n*. The difference (between 2 and 3) is a constant which cannot be distinguished by a complexity argument of this sort. It is an empirical question which is preferable.

(77) "Obviously, acceptability will be a matter of degree, along various dimensions. One could go on to propose various operational tests to specify the notion more precisely (for example, rapidity, correctness, and uniformity of recall and recognition, normalcy of intonation). For present purposes, it is unnecessary to delimit it more carefully." [Chomsky65 pp. 10]

We are still experimenting with the YAP system, looking for a more complete solution to the closure puzzle.[54]

2.5.1 Predictions

Many of Frazier's observations also apply here because YAP closes late by default as in her model (except for the A-over-A early closure principle). As long as A-over-A early closure doesn't apply, YAP behaves just like Frazier's model. In particular, both Frazier's late closure and YAP are not premature, unlike Kimball's scheme. Consider the "counter-example" to Kimball's early closure:

(78) I called the guy [who smashed my car ... up.]
(79) I called the guy [who smashed my car] ... a rotten driver.

Kimball's scheme prematurely closes the root clause just before *who* which is not an immediate constituent of the root. That is, it prematurely decides the root looks like $[_S$ I called NP] regardless of what follows *who*. As we have previously noted, when *a rotten driver* is finally reached, Kimball's scheme will be stuck. Frazier's late closure is an improvement because it keeps the root open until *a rotten driver* is parsed. YAP behaves just like Frazier's model in this case, because the A-over-A early closure principle does not apply. Hence, YAP is not premature (at least in this case).

54. The A-over-A closure principle is an incremental forgetting procedure. One could imagine another type of forgetting procedure which waited until the system ran short on space and only then it would search the stack for "garbage". (In some sense, Frazier's PPP avoids "shunting" until it is running short on space. Hence the PPP is *effective*, though the SSS is now stuck with the problem.) In this framework, the forgetting procedure acts as a background process which "interrupts" the parser whenever space runs short. This interrupt approach is quite plausible though it poses a few problems. First, like a LISP garbage collector which also waits until the computer is out of CONS space, it is not quasi-real-time (bounded amount of time between reading any two input symbols). This is a particularly undesirable property of LISP for real-time applications (like airline guidance systems) because the airplane might crash during a garbage collection. Secondly, interrupt driven systems are extremely difficult to debug and verify because it is very difficult to replicate the same situation twice. Consequently, it would appear quite difficult to model real psychological data within an interrupt framework. Thirdly, the interrupt mechanism is yet another device which must be stipulated. The incremental approach avoids all of these technical problems.

(80) John said [$_1$ that Bill said [$_2$ that Sam said [$_3$ that ...

YAP's closure is more <u>effective</u> in the right branching case because A-over-A early closure will apply. For example, pure late closure will eventually lead to a memory overflow in right branching sentences like (80). Pure late closure will find right branching just as bad as center-embedding. On the other hand, YAP's early closure will constantly close nodes early (before reading the entire sentence), thus preventing a memory overflow. For example, it will close s_1 ([$_1$ that Bill said S]) as soon as it attaches the last daughter to s_2. In sentences like (80), early closure removes nodes just as fast as new ones are being formed. In this case, YAP is <u>effective</u> (unlike Frazier's late closure).

(81) John said [$_1$ that Bill called the guy [$_2$ that Sam said [$_3$ that ... X
(82) #John said [$_1$ that Bill called the guy [$_2$ that Sam said [$_3$ that ...]] X

There are some empirical consequences of closing early. For example, nothing can attach to a closed node. Hence it should be possible to test the A-over-A early closure principle by noting whether or not nodes closed under the principle actually do block further attachments. For example, in (81) once s_1 is closed it shouldn't be possible attach X to it as in (82). We will illustrate several types of X's: adjuncts, conjuncts and optional arguments.

(83)	John said [$_1$ that Bill called the guy ... yesterday.	*adjunct*
(84)	John said [$_1$ that Bill called the guy ... and Sam called the girl.	*conjunct*
(85)	John said [$_1$ that Bill called the guy ... a rotten driver.	*optional argument*

Closure principles merely state which attachments are possible; they do not specify preferences. There is considerable literature noting that X's tend to attach as low as possible. A similar principle will be discussed in chapter 4. It will be qualified to favor attachments to the lowest possible *open* node.[55]

There is a second testable prediction: no interpretive rule can apply into a closed node. That is, linguistic domains (command, c-command, f-command, etc.)[56] have gaping holes where phrases have been closed off. These holes are opaque islands to rules of bound anaphora (reflexive and reciprocal),[57] quantifier scope,[58] and reference (noncoreference). We will show that the prediction appears to hold for Lasnik's noncoreference rule. We will not discuss other interpretive rules here.

2.5.2 Adjuncts

The underlined phrases in (86) and (87) are called *adjuncts*. They can generally attach to any *open* node along the right hand edge, thus accounting for the multiple interpretations. (Admittedly, there is a strong preference for low attachments.)

55. This will attach to the lower matrix even if the higher attachment is the only grammatical possibility. For example, (a) and (b) are marginal because the final phrase tends to attach low, which is ungrammatical.

(a) ?I looked the guy who smashed my car up.
(b) ?Put the block which is on the box on the table.

It seems that this is the correct prediction in the unmarked case; the acceptability might improve if the parser could be given some helpful hints (punctuation or intonation breaks) to block the low attachment. Unfortunately, this account incorrectly predicts that the sentence will become more acceptable if there is a second argument for the higher matrix as in: *#I looked the guy [who smashed my car up] up.* The second *up* cannot attach to the embedded clause, so it should attach to the higher matrix, fulfilling the grammaticality requirements. Unfortunately, the sentence is much worse with the second *up*. This is a serious problem for the current approach.

56. Interpretive rules, such as Lasnik's Noncoreference rule, apply over limited domains of the parse tree. We have already defined *command*; *c-command* and *f-command* are slight variations. *Command* is defined in terms of clauses, *c-command* in terms of constituents, and *f-command* in terms of functional structure (chapter 5).

57. reflexive: They hit *themselves*.
reciprocal: They hit *each other*.

58. (a) Everyone in this room speaks at least two languages.
(b) At least two languages are spoken by everyone in this room.

Sentence (a) has so-called *wide* interpretation (for all people there are two languages such that each person speaks them); sentence (b) has *narrow* scope (there are two languages such that everyone speaks them). See [VanLehn78].

(86) John said that Bill did it <u>yesterday</u>.
John said [that Bill did it yesterday]. *low attachment*
John said [that Bill did it] yesterday. *high attachment*

(87) John said that Bill did it <u>to get ahead</u>.
John said [that Bill did it to get ahead]. *low attachment*
John said [that Bill did it] to get ahead. *high attachment*

The interesting claim is that adjuncts cannot attach to closed nodes. For example, *yesterday* can not attach to s_1 in sentences like (88) because A-over-A early closure would apply first.

(88) John said [$_1$ that Bill said that Sam said ... that Jack did it yesterday.

Although this seems to be the case, it is very hard to test the constituency relations with time adverbials like *yesterday*. Purpose adjuncts (such as *to get ahead*) suggest a much sharper test. Notice that (89) and (90) have different understood subjects, reflecting the different constituency relations.

(89) John said [that Bill did it (*for Bill*) to get ahead].
(90) John said [that Bill did it] (*for John*) to get ahead.

It is possible to test the constituency relations indirectly using well-known facts about subjects. For example, (91)-(92) are unambiguous; the alternative constituency relations (93)-(94) are ungrammatical since they violate binding conditions on reflexives.

(91) They said [that <u>Bill</u> did it to get <u>himself</u> out of hot water].
(92) <u>They</u> said [that Bill did it] to get <u>themselves</u> out of hot water.

(93) *<u>They</u> said [that Bill did it] to get <u>himself</u> out of hot water.
(94) *They said [that <u>Bill</u> did it to get <u>themselves</u> out of hot water].

Now, it should be possible to test whether a node is closed. The purpose adjunct in sentences (95)-(98) must attach to s_1. But this will be unacceptable when s_1 is closed as in (98). As usual, the borderline cases (96)-(97) are somewhat marginal.

(95) Did you hear [$_1$ they did it to get themselves out of hot water?
(96) ?Did you hear [$_1$ they said that Bill did it to get themselves out of hot water?
(97) ??Did you hear [$_1$ they said that Bill said that Sam did it to get themselves out of hot water?
(98) # Did you hear [$_1$ they said that Bill said that Sam said that Jack did it to get themselves out of hot water?

2.5.3 Conjuncts

There is a similar argument using conjuncts instead of adjuncts. Assuming that closed nodes cannot be conjoined, conjunction should become more and more difficult in (99)-(101), since s_1 is more and more likely to close early.

(99) I saw a boy [$_1$ who dropped the delicate model airplane] and who picked it up and began to cry.

(100) ?I saw a boy [$_1$ who dropped the delicate model airplane [$_2$ he had so carefully been making at the school]] and who picked it up and began to cry.

(101) ??I saw a boy [$_1$ who dropped the delicate model airplane [$_2$ he had so carefully been making at the school [$_3$ where I went [$_4$ when I was young]]]] and who picked it up and began to cry.

The claim that conjunction is limited to open nodes is also useful in parsing.[59] Suppose that we had an algorithm for deciding closure. Then we would know exactly which conjunctions are possible because conjunction is permitted between open nodes of the same category.[60] This considerably reduces the combinatoric explosion of possibilities that has made it so troublesome to parse conjunctions. It is an interesting fact that conjunctions, at least their acceptable interpretations, are never more than a few ways ambiguous, even though non-deterministic parsers (such as ATNs) can often find quite a number of absurd possibilities.

59. Some open nodes may not permit conjunction because they are stacked up and hence out of view until the lower possibilities fail. The preference for the lowest open node will be discussed in chapter 4.
60. There are some grammaticality constraints on conjunction which further restrict the possibilities which become important to chapter 9.

2.5.4 Optional Arguments

There is a third argument supporting early closure. Unfortunately the data are extremely controversial[61] and there may be several alternative accounts for the facts. It would not be disastrous for the A-over-A early closure principle if the facts happen to fall the other way. Nevertheless, we will give the argument because it illustrates the approach, even though the evidence is not as clear as it might be.

(102)-(105) test whether s_l is open or closed. If it is closed, the optional argument *a rotten driver* cannot attach and hence the sentence should be unacceptable. This accounts for the judgments in the extreme cases; s_l is open in (102) and hence (102) is acceptable, whereas s_l is closed in (105), and hence (105) is unacceptable. As usual, the borderline cases (103)-(104) are marginal. The A-over-A early closure principle happens to exclude these marginal cases here; this should not be taken too literally.

(102) Did you hear [$_1$ that I called the guy [$_2$ who smashed the car]] a rotten driver?

(103) ?Did you hear [$_1$ that I called the guy [$_2$ who smashed the car [$_3$ that I bought last year] a rotten driver?

(104) ??Did you hear that I called the guy who smashed the car [$_3$ that I bought last year [$_4$ just after the old one needed a new transmission]] a rotten driver?

(105) #Did you hear that I called the guy who smashed the car [$_3$ that I bought last year [$_4$ just after the old one needed a new transmission [$_5$ which would have cost $100]]] a rotten driver?

This account crucially depends on the optionality of the argument *a rotten driver.* If it were obligatory, then it wouldn't be possible to close s_l until a second argument to *call* is found. And hence, early closure would not be an adequate account of the data because it cannot apply to the crucial matrix s_l.[62] There is some evidence that *a rotten driver* is optional; our informants report that (105) is much better without the final phrase. (This judgment is controversial.)

61. Berwick (personal communication) reports different judgments when the crucial examples were spoken. Our own informants were given written texts. Both experiments were informal.

62. A very plausible alternative is that *call* is lexically ambiguous; there is a verb *call NP* as in *I called John* and there is another verb *call NP NP* as in *I called him a name.* Assuming that a clause can't be closed until its predicate has been disambiguated, early closure cannot apply to the crucial matrix containing the verb *call.* And hence, the data may not be relevant to the closure issue. One could take this argument to an extreme and say that all verbs are lexically ambiguous and cannot be disambiguated until the clause is completely parsed, and hence, early closure would always be blocked. Then it isn't clear how right branching sentences could be parsed. The lexical ambiguity argument is very tricky.

(106) Did you hear that I called the guy who smashed the car that I bought last year just after the old one needed a new transmission which would have cost $100?

2.5.5 Noncoreference

Lasnik's noncoreference rule [Lasnik76] is another source of evidence. Previously we showed that noncoreference in sentences like (107)-(109) was less and less likely to apply. In this subsection, we will claim that noncoreferential links cannot be established into a closed node. Again, the extreme cases are much sharper than the borderline. Noncoreference is clearly established in (107) where the crucial clause is open. The judgments become less and less sharp as s_1 is less and less likely to be open.

(107) *Did you hear [$_1$ that John$_i$ told the teacher that John$_i$ threw the first punch?
(108) ?Did you hear [$_1$ that John$_i$ told the teacher [$_2$ that Bill said that John$_i$ threw the first punch?
(109) Did you hear [$_1$ that John$_i$ told the teacher [$_2$ that Bill said [$_3$ that Sam thought [$_4$ that John$_i$ threw the first punch?

2.5.6 Root Clauses

The A-over-A closure principle (unlike Kimball's account) predicts that root clauses have a special status with respect to closure. The root clause will never close because it can't have a mother. In particular, this suggests that it is always possible to conjoin to the root no matter how many clauses intervene.

(110) I saw a boy who dropped the delicate model airplane he had so carefully been making at the school where I went when I was young and you saw a girl do the same.

Similarly, this predicts that root clauses can always take adjuncts. However, it does not predict that optional arguments can attach to the root because they are dominated by a verb phrase which does have a mother. Hence, the verb phrase could close early, blocking optional arguments from attaching.

(111) #Joe [$_{vp}$ figured [that Susan wanted to take the train to New York] ... out.

2.6 Summary

In conclusion, we have argued that a memory limitation reduces the overall time and space requirements (by fiat); the competence model alone cannot achieve such tight bounds. Although it is very difficult to discover the exact memory allocation procedure, it seems that the closure phenomenon offers an interesting set of evidence. There are basically two extreme closure models in the literature, Kimball's early closure and Frazier's late closure. We have argued for a compromise position, the A-over-A early closure principle, which shares many advances of both previous proposals without some of the attendant disadvantages. Our principle is not without its own problems; the borderline cases are extremely difficult. It seems that there is considerable work to be done.

3. Constituent Structure Implementation

YAP's searches for a mapping from a string of words to a set of grammatical relations (subject, object, etc.) In the Bresnan-Kaplan system [Kaplan and Bresnan80], the resulting grammatical relations form a functional structure (fstructure). There is an intermediate representation called the constituent structure (cstructure) which captures structural relations (mother, daughter, sister, etc). The system has an algorithmic procedure for building the fstructure from the cstructure. The chapter 5 will describe how YAP does this; this chapter is mainly concerned with building the cstructure in the first place.

The cstructure has similar counterparts in most other linguistic representations. For example, transformational grammar starts with a set of CF base rules and then applies a number of transformations. Similarly, ATNs start with recursive transition networks (RTNs) which are CF equivalent and then add a number of augmentations. Bresnan's cstructure is a CF description. The mapping from the cstructure to the fstructure is analogous to transformations in TG and augmentations in ATNs.

There are interesting differences between all these systems; we have adopted the Bresnan-Kaplan framework because it seems easier (to us) to map it into a practical FS deterministic parser. Even if TG, ATNs, and the Bresnan-Kaplan framework are all notational variants of one another (which is unlikely), the Bresnan-Kaplan framework might be more useful for our purposes since it is not obvious how to encode the other models into a FS deterministic parser.[63]

63. There have been many articles relating ATNs to processing strategies [Kaplan72] [Wanner and Maratsos78] [Bresnan78]. All of these require more resources (memory and backup) than we are willing to allocate. Their approach appears to be very difficult; although there was great hope in the early papers, it is very difficult to make further progress. McDonald (personal communication) has pointed out that traditional ATNs are analogous to PLANNER [Hewitt72]; both replace knowledge with brute force automatic backup. More recent AI problem solving languages (e.g. TMS [Doyle78]) replace notions like automatic backup with dependency directed backup. We see the same trend in language processing (e.g. GSP [Kaplan75]) though there are many details to be solved. We have avoided many of these difficult problems by stipulating FS and Determinism. It seems that the Bresnan-Kaplan framework is more compatible with these stipulations than more general frameworks (which permit non-deterministic side-effects), though it would be difficult to formalize this intuition.

(112) I am a boy.
(113) $[_s [_{np-} I] [_{vp} [_v am] [_{np-} [_{np} [_{det} a] [_n boy]]]]]$

This chapter will discuss how YAP builds the cstructure. The problem is to map a sentence like (112) into a structure like (113). The cstructure is a tree[64] of phrases (nodes). Phrases are delimited by square brackets ([ ])[65] labeled with a *category* (part of speech). A category has two parts: a major categorial feature (n, v, a, p)[66] and a "bar" level. YAP uses a three-bar system; there are nouns (n), noun phrases (np) and np bars (np-). Similarly there are verbs (v), verb phrases (vp), and participial phrases (vp-).[67] In all, YAP has 4 major categorical features which have three bar levels, forming 12 categories. There are 6 other categories: s, s-, det, comp, conj and punct.[68]

3.1 The Machine State

YAP has four components taken almost directly from Marcus' Parsifal:

<u>YAP</u>	<u>Parsifal</u>
(114) the input stream	the input stream
(115) the upper buffer	the stack
(116) the lower buffer	the lookahead buffer
(117) a deterministic FS control device	a grammar of production rules

A snapshot of the machine is shown in figure 1. The string "= =WALL= =" is printed between the upper and lower buffers. Buffer cells are filled with nodes (parsed phrases) which are printed in square brackets ([ ]).[69] Both buffers grow in toward the WALL as the machine parses toward the center (the WALL) from both directions (both top-down from the root and bottom-up from the input). The upper buffer contains mothers which are building down to the WALL and the lower buffer contains daughters building up

64. This condition will be weakened to encode structural ambiguity (pseudo-attachment).
65. These are often called *phrase markers* (or *P-markers*) in the linguistic literature.
66. n = noun; v = verb; a = adjective; p = preposition
67. The "bar" system was first introduced in [Chomsky70] to describe certain generalities between noun phrases and clauses (i.e. *John's having criticized the book* and *John has criticized the book*). See [Jackendoff77] for a more current reference. The term *projection* refers to the next higher bar level. For example, *np-* is a projection of *np* and *np* is a projection of *n*. The third bar level is the *maximal projection* (in YAP). (There have been proposals for five bar systems.)
68. s = sentence; s- is a projection of s; det = determiner; comp = complementizer (*for, that, ...*); conj = conjunction; punct = punctuation. These categories don't fit the bar pattern very well.
69. Printing is a very expensive task; it requires searching the fringe of the parsed subtrees to find the individual words. The parser itself is not permitted to undertake such expensive tasks. Technically the printer is not part of the machine.

Fig. 1. A Snapshot
sentence: I am a boy.
input pointer: boy.

up3		
up2		
up1	$[_{s}]$	*the upper buffer*
	==WALL==	
down1	$[_{np\text{-}}$ I]	*the lower buffer*
down2	$[_{v}$ am]	
down3	$[_{det}$ a]	

to the WALL. Nodes (parsed phrases, i.e. nonterminals) enter and exit near the WALL in stack fashion (via *push* and *pop* operations). That is, up1 and down1 are the "top" of their respective buffers (stacks). New words (terminals) enter the lower buffer from the "bottom" (down3).

YAP is deterministic and FS for reasons discussed previously. The control device (117) is defined to be deterministic. That is, from any machine state there is exactly one applicable grammar rule; backup is absolutely excluded. The FS limitation has been implemented in YAP by truncating the buffers to a fixed length and limiting the size of a buffer cell. The bounds on the two buffers have not yet been defined. The first three cells of each buffer are referenced so frequently that it is convenient to name them *up1*, *up2*, ..., *down3* as in figure 1. In fact, the buffers may be longer. The complexity arguments suggest that they should be limited, but it is not clear what the limits should be.[70] Setting the exact limits (constants) would require considerable psychological experimentation. The length of the upper buffer measures the maximum allowable depth of center embedding. The lower buffer measures the degree of lookahead.[71]

70. The bound on each buffer is a parameter which can be adjusted at runtime.
71. Chomsky (personal communication) points out that bounded lookahead might be equivalent to some sort of bounded backtracking. In which case, the lower buffer could be thought of as measuring the degree of backtracking. [Ullman65] discusses two interesting formalizations of bounded backtracking. ("Bounded parallelism" is probably very similar to bounded backtracking and bounded lookahead.)

There are some interesting issues associated with fixing the size of nodes. In Parsifal, a node is literally a pointer to a subtree (parsed phrase). A YAP node is an abstraction of the relevant features, not the entire tree itself.[72] It is important to bound the size of a node in order to prevent encoding unbounded memory into the nodes.[73] This guarantees that any predicate can be tested in a fixed amount of time. Parsifal stores subtrees in the stack cells; it could take an arbitrary amount of time to search a subtree for some property (such as the value of a trace.)[74] Similarly, the formal system outlined in [Kaplan and Bresnan80] permits two unbounded nodes to be unified which also requires unbounded time.[75] Although this is a theoretical point, it does bring up some very difficult questions regarding abstraction (inheritance). Which features does a mother inherit from its daughters and which features are opaque to further inquiry? This question will be studied in chapter 5.

3.2 Production Rules

Only one more component is needed before the machine can run. We have to specify a procedure for determining which actions to apply next. We will begin by describing a very general technique. The next few sections will present some more specific techniques which should cover the most common unmarked cases. The more general techniques will be used only in very marked exceptional situations. We introduce them first because it is relatively easy to see that they are sufficiently powerful; however, they are so powerful that it is very difficult to combine them effectively into a good structured program (grammar).

The general technique is to use a set of production rules (like Parsifal's grammar rules) which uniquely determine the actions to perform. That is, the first applicable production rule is selected. We are strongly depending on Marcus' Determinism Hypothesis; the rules cannot backup or sprout new processes like a non-deterministic machine. A sample rule might be:

72. Actually the tree structure is maintained for the printer's convenience; the parser itself does not look beyond a single level of tree structure. The parser is a FS transducer which inputs words and outputs tree structure. These structural links, which are maintained for the printer, should be viewed as part of the output, not the internal state.
73. If a node could be arbitrarily large then it could encode anything. *Godel-encoding* is a particularly extreme technique of accomplishing this undesirable consequence.
74. *Traces* are a formal linguistic object which will be discussed in chapter 8. Parsifal allows traces to be bound to other traces and hence it may require unbounded time to retrieve a value from a long chain of traces.
75. This property provides considerable computational power. That system is capable of parsing CS languages of the form: $a^n b^n c^n$ [Kaplan (personal communication)].

```
(118) (defrule attach-subj
          (pattern (=s) (=np- =vp))
          (action (attach)))
```

This rule would say, if there is an *s* (sentence) node just above the wall (up1) and there is an *np-* followed by a *vp* just below the wall (in down1 and down2), then attach the *np-* (down1) to the *s* (up1). It is very similar to a CF rule of the form:[76]

```
(119) s -> np- vp ...
```

The pattern has a limited window; it can only reference the first three cells in each buffer and features immediately attached to them. (120) lists the syntax for a pattern. There are six predicates <up1>, ..., <down3> associated with up1, ..., down3, respectively.[77] If the predicates and the lisp expression[78] return true, then the pattern "matches".[79]

```
(120) (pattern
          (<up3> <up2> <up1>)
          (<down1> <down2> <down3>)
          <lisp expression>))
```

76. CF rewrite rules are often viewed as top-down (generative). This asymmetry is purely a matter of convention; they could have been formulated in a bottom-up fashion. Our representation is neutral with respect to top-down and bottom-up.

77. If the pattern contains less than three predicates, the specified predicates apply to the cells closest to the WALL. For example, (118) applies to up1, down1 and down2 because up1 is the closest to the WALL from the upper buffer, and down1 & down2 are the closest from the lower buffer.

78. This lisp expression must be side-effect-free (cannot change the state of the machine in any way). It is also constrained to the first three cells in each buffer and their immediate descendants (by convention).

79. Although it is useful to think of the pattern matching as a linear search, they are actually inverted (hashed) on <up1> and <down1>. In practice, approximately seven patterns are tested before finding a match. The test/match ratio had been 4:1 before certain rules were added. Theoretically it should be possible to do much better; the test/match ratio should be 2:1 or better.

The matching procedure deserves much more attention. This may be the proper place to incorporate lexical expectation and extremely subtle preference data, which are often taken as evidence for a backup mechanism. Since lookahead is analogous to backup, it ought to be possible to encode these facts in a lookahead framework.

3.3 Actions

The grammar rules (the deterministic FS control device) modify the machine state through a number of actions. This section will discuss three primitive actions: attach, predict and close. These basic operations appear in most parsers in one way or another. In an ATN the corresponding operations traverse an arc, push to a new network, and pop from a network. In Earley's algorithm [Earley70] the corresponding operations are scan, predict and complete. Basically, any tree traversal algorithm will have three corresponding operations:

(121) move *across* from one sister to the next	(attach, traverse arc, scan)
(122) move *down* from a mother to the first daughter	(predict, push)
(123) move *up* from the last daughter to the mother	(close, pop, complete)

These actions are implemented in terms of buffer operations. Recall that both buffers are building toward the WALL; the upper buffer holds mothers looking top-down for daughters (on the other side of the WALL) whereas the lower buffer holds daughters looking bottom-up for mothers. (The grass is always greener on the other side of the WALL, so to speak.)[80] When a daughter and a mother finally find each other (attach), the daughter is popped from the lower buffer because it is no longer looking for a mother. It is then pushed onto the upper buffer, to enable it to find some of its own daughters.[81] As we will see, up1 will inherit certain features (e.g. person, number, gender, ...) from down1 to reflect the attachment. Finally, the mother and

Fig. 2. The Attach Action
Attach pops down1 from the lower buffer and pushes it into the upper buffer.

sentence: I am a boy.
input pointer: boy.

before	after
$[_s]$	$[_s$ I]
==WALL==	$[_{np-}$ I]
$[_{np-}$ I]	==WALL==
$[_v$ am]	$[_v$ am]
$[_{det}$ a]	$[_{det}$ a]

80. In GSP terminology [Kaplan75], the upper buffer holds *producers* and the lower buffer holds *consumers*.
81. Daughters can be attached before they themselves are complete. This is crucial for early closure.

daughter are linked together in the output structure. This link is also available to the printer, and hence, up1 prints differently after the attachment. For example in figure 2, up1 is printed as $[_S\,]$ before the attachment because it dominates no words, but afterwards, it is printed as $[_S$ I] because it dominates the word *I*.

The machine proceeds in a middle out fashion away from the WALL. First, it tries to attach down1 to up1 as we have just seen. If that fails, it starts a new node between up1 and down1. This is the *predict* operation. For example, suppose that YAP finishes parsing the subject in figure 2 by some yet unspecified means, leaving the machine state ready for the predict action as in figure 3. Up1 contains a clause ($[_S$ I]) looking for a *vp* daughter and down1 contains a verb $[_V$ am] looking for a *vp* mother. The predict action starts a *vp* node between up1 and down1 to bridge the gap. The machine can now continue by attaching up1 to down1 just as it did in figure 2.

YAP will continue predicting and attaching until it reaches the state specified in figure 4. At this point, up1 is complete. The machine will *close* up1 by popping it from the upper buffer, thus removing it from memory. It cannot take on any more daughters.

3.4 The Phrase Structure Component

Marcus' machine has a number of production rules like (121) to decide which actions to perform. It would be possible to write a complete grammar in this form. If we did so, YAP would look very much like his machine. The problem with writing a grammar in production rules is that the performance and the competence

---

Fig. 3. The Predict Action
Predict will start a new node between up1 and down1.

sentence: I am a boy.
input pointer: .

before	after
$[_S$ I]	$[_S$ I]
= = WALL = =	= = WALL = =
$[_V$ am]	$[_{VP}\,]$
$[_{det}$ a]	$[_V$ am]
$[_n$ boy]	$[_{det}$ a]
	$[_n$ boy]

Fig. 4. The Close Action
Close pops the upper buffer, thus removing up1 from memory. Nothing more can attach to this *np*.

sentence: I am a boy.
input pointer:

<u>before</u>	<u>after</u>
$[_s$ I am a boy]	$[_s$ I am a boy]
$[_{vp}$ am a boy]	$[_{vp}$ am a boy]
$[_{np-}$ a boy]	= = WALL = =
= = WALL = =	$[_{punct}$.]
$[_{punct}$.]	

components tend to become tangled; it is very difficult to write a good structured program (grammar) with such elementary building blocks. [Swartout78], [Shipman78] and [Marcus79, chapter 4] have observed that there are phrase structure (PS) rules hidden inside Marcus' grammar. Shipman then wrote a PS machine which used phrase structure rules to decide when to activate rules.[82] It would be desirable if we could add phrase structure rules to YAP so that it could select the next action in an orderly way. The phrase structure component should cover most normal unmarked cases; production rules are reserved for marked exceptions. A typical YAP phrase structure rule is as follows (omitting details):

```
(124) (def-ps-rule finite-s s
          (csubj obl (s- np-))
          (chead obl (vp)))
```

This ps-rule is similar to the two CF rules:[83]

82. Marcus has a notion of active rules. For technical reasons, we have not incorporated this idea explicitly in YAP. The notion is in fact implicitly encoded in nodes. (Each node knows what it is looking for.)
83. Technically, it is closer to the following CF rules. However, the nonterminals (csubj, chead, ...) are always non-branching.

```
s -> csubj chead
csubj -> s-
csubj -> np-
chead -> vp
```

(125) s -> np- vp
(126) s -> s- vp

In English the rule says that a *finite s*[84] has two obligatory daughters; the first is the surface subject (csubj) and the second is the head (chead). The first can be either an *s-* or *np-*, and the second, a *vp*. This rule could have been written as a large number of marked production rules; the ps rules are more perspicuous. For example, a single ps-rule replaces ten of Marcus' rules for parsing auxiliaries.[85] See [Shipman78] for a translation procedure from ps-rules to Marcus' production rules.

There is a PS pointer associated with each node to indicate what the node is "looking for". A PS pointer is written in dotted rule notation where the dot (.) marks which terms have been parsed (see figure 5). The PS pointer is automatically advanced when a daughter is attached as in the figure.[86]

YAP will use the PS rules to select the next action. When there are no applicable marked rules, the interpreter tries to apply the PS rules. There are three possible PS actions: ps-attach, ps-predict, and ps-close. In YAP they are implemented as follows:[87]

(127) ps-attach: If down1 can attach to up1, then do so. (figure 5)
(128) ps-predict: If the category of up1's next daughter can be determined, then predict a node of that category. (figure 6 top)
(129) ps-close: If up1 can be closed, then do so. (figure 6 bottom)

84. A *finite clause* is tensed, as opposed to an infinitive or participial phrase.

finite:	I am a boy.
infinite:	To be a boy is tough.
participial:	Being a boy, I know how he must feel.
	Raced past the barn, the horse felt like getting even.

85. Auxiliary verbs are "helping" verbs such as: *be, have, will, can, do ...*

86. Parentheses () denote optional terms, brackets {} denote exclusive disjunction, and * is the Kleene star for arbitrary repetition. Brackets have very restrictive distribution since they are difficult to express within the determinism framework.

87. *ps-predict* has a top-down asymmetry which is very unfortunate. To compensate for this deficiency, there are quite a number of production rules to predict bottom-up. The grammar would be much simpler if the ps-predict rule were more symmetric.

Fig. 5. PS Attach
sentence: I am a boy.
input pointer: boy.

$[_{s}\ ]$	finite-s -> . csubj chead
= = WALL = =	
$[_{np\text{-}}\ I]$	normal-x -> cword .
$[_{v}\ am]$	normal-x -> cword .
$[_{det}\ a]$	normal-x -> cword .

Because down1 is a possible *csubj* for up1's *finite-s*, the default PS-attach rule will attach down1 to up1, leaving the machine in the following state. Notice that the PS pointer associated with the *s* node is automatically advanced.

$[_{s}\ I]$	finite-s -> csubj . chead
$[_{np\text{-}}\ I]$	normal-x -> cword .
= = WALL = =	
$[_{v}\ am]$	normal-x -> cword .
$[_{det}\ a]$	normal-x -> cword .

All these rules are depended upon the ps pointers; the conditionals (can attach, can predict, and can close) are functions of the ps pointers. These rules are the defaults which can be over-ruled in the marked case by a production rule. By introducing these ps rules we have greatly reduced the number of marked productions rules. The current grammar has 12 ps-rules and 69 production rules. In practice, the ps-rules and production rules are executed about equally often. The PS rules were designed to strongly resemble Bresnan-Kaplan's constituent structure component just as the production rules resemble Marcus' grammar.

3.5 Ordering PS Actions

(130) attach
(131) predict
(132) close

The ps rules have an <u>unmarked</u> order (130)-(132) which can be over-ruled by a *marked* production rule. In the unmarked case, first try to attach down1 to up1. If that doesn't work out, then try to predict. Finally, try closing up. Empirically, this order seems to require a minimum number of marked rules. It favors attaching early (low) and closing late. Late closure was discussed in chapter 2; early attachment is the subject of

Fig. 6. PS Predict & PS Close
PS Predict
sentence: I am a boy.
input pointer: .

[$_{s}$ I]	finite-s -> csubj . chead
= = WALL = =	
[$_{v}$ am]	normal-x -> cword .
[$_{det}$ a]	normal-x -> cword .
[$_{n}$ boy]	normal-x -> cword .

Since the category up1's next daughter is unique (it must be a *vp*), the PS-predict rule will start a *vp* node in down1, as illustrated below.

[$_{s}$ I]	finite-s -> csubj . chead
= = WALL = =	
[$_{vp}$]	normal-vp -> . chead (cobj) (cxcomp)
[$_{v}$ am]	normal-x -> cword .
[$_{det}$ a]	normal-x -> cword .
[$_{n}$ boy]	normal-x -> cword .

PS Close
sentence: I am a boy.
input pointer:

[$_{s}$ I am a boy]	finite-s -> csubj chead .
[$_{vp}$ am a boy]	normal-vp -> chead (cobj) . (cxcomp)
[$_{np-}$ a boy]	normal-np- -> (cspec) chead .
= = WALL = =	
[$_{punct}$.]	normal-x -> cword .

Since up1 can close, the PS close operation would pop it from the upper buffer, thus removing it from memory, so no further attachments can be considered.

[$_{s}$ I am a boy]	finite-s -> csubj chead .
[$_{vp}$ am a boy]	normal-vp -> chead (cobj) . (cxcomp)
= = WALL = =	
[$_{punct}$.]	normal-x -> cword .

chapter 4. The rule ordering would attach *X* as low as possible in structures like (133) because *ps-attach* precedes *ps-close*. (134)-(136) illustrate this for adjuncts, conjuncts and optional arguments, respectively. The next chapter will compare this approach with alternatives in the literature.

(133) John called the guy who called the girl who called ... X

(134) John called the guy who called the girl who called ... yesterday. *adjunct*
John called the guy who called the girl who called ... to make {himself, herself} feel better.

(135) John called the guy who called the girl who called ... and said "hello". *conjunct*

(136) John called the guy who called the girl who called ... a rotten driver. *optional arguments*
John called the guy who called the girl who called ... up.

4. Attachment Strategies

What types of information should drive the attachment process? There are four basic strategies in the literature:[88]

(137) <u>Structural Bias</u>	[Kimball73, 75], [Frazier and Fodor78], [Marcus79], YAP
(138) <u>Lexical Expectation/Arc-Ordering</u>	[Fodor78], [Bresnan78], [Kaplan72], [Wanner78, 79]
(139) <u>Length Bias</u>	[Frazier79], [Frazier and Fodor78], [Fodor and Frazier80]
(140) <u>Semantic Bias</u>	[Crain79]

Although there are valid arguments for each of these positions, we will concentrate on the structural biases in this chapter. YAP can encode the other biases using marked rules.[89] The structural bias is provided (in the unmarked case) by the proposed rule ordering (attach, predict, and then close). It appears very similar results are produced by Frazier's two principles: <u>minimal attachment</u> and <u>late closure</u>. This idea was inspired by [Wanner79 pp. 12] which relates Frazier's principles to certain ATN actions (traverse arc, push and pop) which are similar to our three primitive actions (attach, predict and close.)

88. Few papers fit the categories perfectly. For example, we have listed the Sausage Machine in two places because it has some structural components (minimal attachment and late closure) and some length biases (Preliminary Phrase Packager). Similarly, we could have listed the arc-ordering papers under several headings because arc-ordering can encode many types of biases, as [Wanner79] quite correctly notes.

89. We have very little to say about length biases. Frazier's machine has a front end called the *Preliminary Phrase Packager* (PPP) which segments the input stream into manageable chunks that are "shunted off" to the next higher stage (SSS). The PPP has severely limited memory (about six words) and it has little or no ability to communicate with the SSS except to "shunt" segmented phrases which it will (almost) never see again. This model makes the interesting prediction that preliminary segmentation is subject to length biases.

There are a few problems with this proposal. First off, it is not clear how to build a PPP. Purely bottom-up segmentation is extremely difficult in general, unless one is will to form all possible segments (which is probably not Frazier's intent.) Secondly, although the length biases are certainly real at some level, Frazier's suggestion that they play a major role in parsing is extremely controversial. For example, [Wanner79] observes that the length factor does not appear to alter the preferred interpretation in the following sentences.

Tom said that Bill had taken it out yesterday.
Tom said that Bill had taken it yesterday.
Tom said that Bill took it yesterday.
Tom said that Bill died yesterday.

We will accept Wanner's criticisms of the PPP and his alternative proposal (ordering the actions: attach, predict and then close). The interested reader should investigate his paper for more discussion of the PPP and how it relates to his proposal.

(141) Minimal Attachment: Attach incoming material into the phrase marker being constructed using the fewest nodes consistent with the well-formedness rules of the language.

(142) Late Closure: When possible, attach incoming material into the clause or phrase currently being parsed. [Frazier79 pp. 76]

(143) Minimal attachment = attach before predict
(144) Late closure = close after predicting and attaching

If the two analogies, (143) and (144), are correct, then the proposed unmarked ordering of ps rules is a valid implementation of Frazier's principles. Her principles were designed to capture a large number of performance phenomena, from a psychological point of view. We will address their feasibility from a practical engineering point of view.

4.1 Minimal Attachment

Minimal attachment prefers (146) and (149) because they have fewer brackets (nodes).[90]

(145) The horse raced past the barn (fell). ([Frazier79 pp. 27])
(146) $+[_s [_{np}$ The horse] $[_{vp}$ raced past the barn]] ... fell
(147) $-[_s [_{np} [_{np}$ The horse] $[_s [_{vp}$ raced past the barn]]] $[_{vp}$ fell]]
(148) Tom heard the latest gossip about the new neighbors (wasn't true). (Similar to [Frazier79 pp. 155])
(149) + Tom heard [the latest gossip about the new neighbors].
(150) − Tom heard [[the latest gossip about the new neighbors] wasn't true].

4.1.1 Sensitivity to Phrase Structure Rules

There is a technical problem with this formulation; minimal attachment is extremely sensitive to slight modifications in phrase structure rules; it would be more robust if it counted limiting growth (like a complexity argument), not individual nodes. It is not clear, for example, that her counting argument can be used to distinguish the following [Frazier79 pp. 24].

90. It is useful to further distinguish the acceptable/unacceptable continuum. The plus symbol (+) is used to indicate a more acceptable sentence; minus (−) indicates a less acceptable one.

(151) Sam hit [the girl] [with a book] *high attachment*
(152) Sam hit [the girl with a book] *low attachment*

The first has one fewer node using her phrase structure rules; they have the same number in our analysis. These borderline cases are notoriously difficult; human judgments tend to be unreliable and indecisive. For example, [Wales and Toner76] have found that certain ambiguous structures have little or no bias; both possibilities are about equally probable. This fact is not captured by most attachment strategies which draw very sharp distinctions. Certainly, both Frazier's minimal attachment and our ordering criteria are guilty of this criticism. Later in this chapter, we will discuss a marked rule (pseudo-attachment) to cover the ambiguous case.

(153) $[_{np}$ the girl] $[_{pp}$ with a book]	*Frazier's analysis*	*high*
(154) $[_{np}$ $[_{np}$ the girl] $[_{pp}$ with a book]]		*low*
(155) $[_{np-}$ $[_{np}$ the girl]] $[_{pp-}$ with a book]	*YAP's analysis*	*high*
(156) $[_{np-}$ $[_{np}$ the girl] $[_{pp-}$ with a book]]		*low*

4.1.2 Explanations for Minimal Attachment

Intuitively, the principle appears to conserve computational resources, although the argument has not been completely formalized. [Wanner79] argues that it is generally more efficient to attach before predicting because predictions postulate an additional node which presumably involves a certain additional cost. Hence it is generally cheaper to order attach before predict. This ordering happens to be consistent (more or less) with Frazier's minimal attachment strategy.

It is very difficult to formalize this argument. Although it is *generally* cheaper to attach before predicting, attaching first isn't *always* cheakper. For example, there are structurally ambiguous sentences such as (151)-(152) where attaching first is no more efficient. Even if there were n discrepancies between the ordering criteria and Frazier's principles, it isn't clear which *explains* which. Does the ordering criteria explain the minimal attachment principle or the other way around? Nevertheless, there is an interesting correlation. Despite its problems, we will accept Wanner's account as an *implementation* of minimal attachment (and leave the explanation question unresolved).

[Fodor and Frazier80] suggest another explanation. Suppose the parser builds "several" paths in parallel. The first one to finish "dominates subsequent processing". This provides a nice motivation of minimal attachment; presumably the most minimal path would finish the "race" first since it constructed the fewest number of nodes. Similarly, they could account for the ambiguous case as "a double finish" (although Frazier happens to argue that this particular case is unambiguous [Frazier79 pp. 143]).

One has to be careful with the parallel processing account. If it is taken too literally (each derivation has its own processor), it would trivialize the attempts to limit backup/lookahead (by substituting hardware for backup/lookahead). There ought to be a mechanism for bounding parallelism just as there is a mechanism for bounding lookahead in Marcus-style parsers. (In some sense, backup, lookahead and parallelism are all very similar.) Fodor and Frazier's account would be much more satisfying if they also discussed the limitations of the parallelism.

It has been very difficult to find a deep explanation for the principle because it is heuristic (in our framework). There are several cases where the principle can be overridden. For example, there are the ambiguous cases just mentioned. Also, it has been argued that semantic and pragmatic biases can influence the judgments. Furthermore, there appear to be some empirical constructions where the most minimal attachment is excluded (by competence constraints) permitting a less minimal attachment. These (rare) cases constitute yet another class of exceptions, at least in our framework.[91] It is a heuristic to be applied when there are no reliable clues (semantics, pragmatics, or grammatical constraints). Minimal attachment is not like center-embedding, for example, which is universally unacceptable. Center-embedding can be explained by the FS hypothesis; we are not likely to find a similar explanation for minimal attachment. It is a "least effort" heuristic (in linguistic terminology, it is a "markedness" principle). Heuristics are generally more difficult to explain than universals like center-embedding.

91. Actually Frazier (personal communication) disputes this point. Since her machine is non-deterministic, these "exceptional" cases are less problematic: her machine simply takes the most minimal path first and then backs up when it encounters a dead end. Hence it will eventually find the most minimal grammatical interpretation. In our deterministic framework, we have marked rules to look ahead for the problematic cases. In either framework, though, these exceptional cases pose a difficulty for an explanation because it is not clear how one can constrain the backup/lookahead mechanism.

4.1.3 Left Branching Structures

There are some cases where the heuristic is crucial. For example, extreme non-minimal attachment (predicting before attaching) fails on a left branching structure such as (157), where it would predict infinitely many noun phrases.[92] Although the most extreme non-minimal position is theoretically inadequate, there are many compromise positions which may suffice. For example, a parser could make a few predictions before attaching, thus creating slightly non-minimal structures without the theoretical inadequacies. There is no explanation for the *most* minimal strategy.

(157) np -> np's n
(158) John's father's ... brother's friend

4.2 Garden Paths

Left branching is an extreme case; Frazier's experiments were more concerned with the well-known garden path (GP) phenomena such as (159)-(162). These are called GP sentences because the reader is led down the garden path so to speak. It would appear that the performance model has optimized the process of recognizing the vast majority of sentences which do not contain garden paths so that these GP sentences are no longer acceptable.

(159) #The horse raced past the barn fell.
(160) #The ship floated on the water sank.
(161) #John lifted a hundred pound bags.
(162) #I told the boy the dog bit Sue would help him.

92. Some parsing models in the literature actually have this problem. For example, the LL(k) algorithm, which predicts before attaching, will infinitely predict on left branching structures. Also, the Harvard *Predictive* Analyzer (HPA) [Kuno66] ran into difficulties because it predicted first. They invented the *shaper* heuristic to prevent the machine from predicting more terminals than there were input symbols. Needless to say, it is possible to do much better by attaching sooner as in Earley's Algorithm [Earley70]. A well-formed substring (WFSS) table [Kuno and Oettinger63] would also solve the problem, though it requires unbounded space. (It could be argued that the WFSS provides the necessary bottom-up information by constraining the search space as it does.)

The GP interpretations result from attaching at the critical point instead of predicting. For example, the machine will prefer to attach in (163), thus taking the first fatal step down the garden path. The grammatical (but unlikely) interpretation requires predicting a clause node instead of attaching.

(163) [$_s$ I told the boy the dog bit]
[$_s$ the dog bit]
[$_{vp}$ bit]
==WALL==
[$_{np-}$ Sue]
[$_v$ would]
[$_v$ help]

The "non-minimal" interpretation can be forced in the presence of positive evidence.[93] For example, (165) is acceptable because there is sufficient positive information (an unambiguous +*en* morphological feature) to predict a reduced relative clause[94] when the machine is in state (166). On the other hand, sentence (164) does not have the same reliable evidence for a reduced relative, and hence there is insufficient motivation to predict the additional node.[95] Since the *vp* can't attach to [$_{np-}$ the horse] without the reduced relative node, and the reduced relative node can't be predicted, the machine will ps-close, the only ps-action left. In this

93. In our formulation the positive information will be in the limited lookahead buffer; in Frazier's model, it will be discovered by the limited backup mechanism.
94. The terminology, *reduced relative*, comes from an old deletion analysis which derived (b) from (a) by deleting *who was*.

(a) the horse who was taken past the barn
(b) the horse taken past the barn

This construction has also been called *whiz* deletion (short for *who is* deletion). Instead of deleting, YAP base generates the construction directly as follows: [$_{np-}$ the horse [$_{vp-}$ taken past the barn]]. In this analysis, predicting the relative clause amounts to predicting the *vp-* node.
95. If YAP looked sufficiently far ahead, it would find sufficient evidence for the reduced relative. We are assuming that one generally doesn't look that far ahead.

case, closing is the first fatal step down the garden path.[96]

(164) #The horse raced past the barn fell.
(165) The horse taken past the barn fell.

(166) $[_s$ The horse]
$[_{np-}$ The horse]
==WALL==
$[_{vp}$ taken past the barn]
$[_v$ fell]
$[_{punct}$.]

It would be possible to parse garden paths if one looked sufficiently far ahead. Figure 7 illustrates a very marked rule to do so. We assume that most people do not look so far ahead because they have not seen enough evidence to justify the effort. Perhaps, psycholinguists, with their unusual background, have acquired a rule like the "horse-racing" rule in figure 7.[97]

These garden paths should be distinguished from center-embedding because we believe no one (not even the best psycholinguist) can learn to parse deeply center-embedded sentences in real time. Although it would be possible to add a marked rule to parse garden paths, the machine is fundamentally incapable of parsing

96. Frazier's account differs slightly because she uses alternative phrase structure rules.

np -> np vp (Frazier)
$[_{np1}$ $[_{np2}$ the horse] $[_{vp}$ raced past the barn]]
np- -> np vp- (YAP)
$[_{np-}$ $[_{np}$ the horse] $[_{vp-}$ $[_{vp}$ raced past the barn]]]

We have attributed the problem to predicting the reduced relative node (*vp-*); in her framework, the problem is to predict the *np1*. The accounts are very similar (modulo the phrase structure rules). In both cases, the machine fails to predict the reduced relative because there is insufficient evidence.

97. Similarly, it is possible to write marked production rules in YAP which violate well-known grammatical constraints such as Ross' Complex Noun Phrase Constraint (CNPC). Although most normal people have extreme difficulty parsing violations of CNPC, there are some experienced linguists who cannot trust their own intuitions because they can parse certain violations with relative ease. Since there are some people (e.g. experienced linguists) who can parse certain violations, a parser should also have this capability although it may require some very highly marked rules. This position is somewhat different from [Marcus79], where it is assumed that the parser should be incapable of violating certain grammatical constraints.

Fig. 7. A Marked Rule to Parse a GP
If YAP had a rule like the ad hoc "horse-racing" rule below, it could parse, *The horse raced past the barn fell.* Of course, there is no evidence that such a rule exists. (This rule also has quite a number of other problems which will not be discussed.)

sentence: "The horse raced past the barn fell."
input pointer:

$[_s$ The horse]
$[_{np-}$ The horse]
= =WALL= =
$[_{vp}$ raced past the barn]
$[_v$ fell]
$[_{punct}$.]

```
(defrule horse-racing
  (pattern (=s =np-) (=vp =v))
  (action (predict 'vp-) (attach)))
```

arbitrarily deep center-embedding. The allowable depth is determined by the limited memory.[98]

4.2.1 Semantic Bias

There is some additional evidence distinguishing the GP case from the center-embedding case. Unlike the center-embedding case, it is possible to reverse the judgments with priming (167) or strong semantic clues [Crain and Coker78] [Crain79] (168)-(173). Non-minimal attachments are generally possible if there is sufficient positive evidence (linguistic training, priming, or semantic clues) to exclude the more minimal interpretations.[99]

98. It is possible to add some marked rules which would occasionally allow an extra level of embedding. Correspondingly, it is possible that a person could learn to recognize an extra level of embedding in many situations. For example, certain experienced psycholinguists have memorized certain 3-deep constructions such as: *#The woman the man the girl loved met died.* However, it is impossible to add enough marked rules to allow arbitrarily deep center-embedding.

99. There is one qualification: the non-minimal attachments are limited to open nodes. Hence semantic biases cannot influence the attachment decisions once a node has been closed. For example, in structures like: [I said $[_1$ you said he said ...X...., *X* cannot attach to s_1 once it is closed, under any semantic context. (Some semantic contexts might block s_1 from closing, and hence indirectly influence attachment decisions.)

(167) There were two horses being raced, one out in the field and the other past the barn. The horse raced past the barn fell. *priming*

(168) The tenant delivered junk mail threw it in the trash. *semantic bias*
(169) #The postman delivered junk mail threw it in the trash.

(170) The cheater furnished the answers passed the test.
(171) #The genius furnished the answers passed the test.

(172) The performer sent the flowers was thrilled.
(173) #The florist sent the flowers was thrilled.

4.2.2 Marcus' Account

This account differs slightly from [Marcus79], where it would be very difficult to state a rule which correctly resolves garden paths, and consequently, his machine will guess which path to take when it cannot correctly resolve the ambiguity. In the garden path case, the machine will take the wrong path. The semantic priming can be explained in the model as reversing the heuristics. Accordingly, we would predict that (174) should be out since the priming has reversed the two paths. The prediction is probably correct.

(174) ?#There were two horses being raced, one out in the field and the other past the barn. The horse raced past the barn.

It is more difficult for Marcus to explain why trained psycholinguists can parse garden paths. Unlike the priming case, the psycholinguist is aware of both paths. If the disambiguating rule cannot be stated, then how is it that psycholinguists seem to parse both of them correctly? It is possible that learning psycholinguistics increases the lookahead buffer, and hence, they can parse certain GPs even though most normal people cannot. However, we have adopted another account. Instead of saying that the GP cannot be resolved by a marked rule, we take the much weaker position that there must be positive evidence to justify the rule. Marcus' position is more restrictive than our own, and hence more theoretically attractive. Unfortunately, in YAP, it was found necessary to enlarge the class of definable rules, and hence, we had to abandon Marcus' position that the "horse-racing" rule (figure 7) cannot be stated, in favor of the weaker position that such a rule is highly marked.

4.2.3 Related Work

This account is somewhat similar to [Bever70] where there was a parsing strategy (175) to account for some of the same empirical facts. We have two slight objections with his strategy: (a) it is not as general as Frazier's formulation, and (b) it conflates performance and competence.

(175) Strategy B: The first N..V.(N).. clause (isolated by Strategy A [which segments clauses]) is the main clause unless the verb is marked as subordinate.

Frazier's minimal attachment also overlaps with [Chomsky and Lasnik77] where some of the same phenomena are described in terms of filters. Frazier's account involves performance whereas filters presumably encode competence. Chomsky and Lasnik say that (176) is ungrammatical; Frazier's principle would imply that it is also unacceptable.

(176) *#The girl saw you is here.

It is very tempting to suggest an explanation. A functionalist might argue that it is ungrammatical *because* it is unacceptable.[100] It is equally mistaken to deduce that unacceptability *follows* from ungrammaticality. A mere overlap between performance and competence does not constitute an explanation (in either direction).[101] On the other hand, the overlap is probably worth studying in more detail. For example, one might look for an explanation in terms of evolution as in [Bever and Langendoen71]. It is unlikely to be pure chance.

100. A functionalist argues that a phenomenon *P* is the way it is because *P* is a necessary by-product of computing some function. In this case, a functionalist might conclude that minimal attachment explains certain ungrammaticality facts *because* certain ill-formed sentences cannot be parsed. This position is taken in [Ades79].

101. Chomsky and Lasnik specifically warn us about certain tempting although incorrect functional "explanations." According to [Chomsky and Lasnik77 pp. 437]. *Similar conclusions are conventional in attempts at functional explanations for properties of physical organs, for example. Thus we can no doubt account for properties of the heart by considering the function of pumping blood, but no one assumes that the embryo decides to develop a heart because it would be useful to have this function filled.* (Most reasonable functional explanations are at the level of evolution. Even if functionalism does not provide an explanation, it is often useful as a motivating force. It may suggest where to concentrate the investigation. Although we are not advocating an extreme functional position, it can be a profitable approach.)

Similarly YAP, which encodes minimal attachment, does not explain minimal attachment or any facts which follow from that (e.g. certain GP phenomena) but merely provides a description. We agree with Frazier's intuition that minimal attachment is a consequence of limited resources. Even if the connection between minimal attachment and limited resources could be proven, we would not have an explanation. It would remain to be seen why people adopt the proposed strategy in favor of some inferior one. Is minimal attachment learned or is it innate as Frazier suggests? These are extremely hard questions; we have only attempted to model (describe) the facts. This work should not be interpreted as an explanation.

4.3 Non-Minimal Attachment

There are a few exceptional cases where the default order (attach, predict, and then close) would produce incorrect results. These exceptional cases should also be a problem for Frazier's principles (which she solves with a backup mechanism.) In our framework, there will be a few marked rules to cover the following exceptions:

(177) early closure (chapter 2)
(178) transformations (chapters 6-9)
(179) non-minimal attachment
(180) pseudo-attachment

Sentences (181)-(186) show that <u>non-minimal</u> <u>attachment</u> is occasionally appropriate. The first sentence in each group is more minimal than the others. It would appear that the parser should not blindly attach without looking ahead at the next constituent for one of these exceptional cases.

(181) I know [the boy].
I know [[the boy] went home]. *null complementizer*

(182) John saw Tom and [Mary].
John saw Tom and [[Mary] saw Sue]. *conjunction*

(183) I told the boy [that].
I told the boy [[that] story].
I told the boy [[that] you liked the story]. *lexical ambiguity*

YAP has marked rules to cover each of these cases. The last group are disambiguated by the *that-diag*, a marked rule to distinguished the various senses of *that*.[102] The first two pairs are disambiguated by a marked rule which predicts an *s* when there is a node looking top-down for an *s* and there is an subject-tense pattern in the lower buffer. For example, an *s* would be predicted in (184).

```
(184) [s I]
      [vp knew]              know-1 -> head . {obj, scomp}
      ==WALL==
      [np- the boy]
      [v went]
      [n home]
```

All of these examples appear to be counter-examples to Frazier's minimal attachment which are easily solved though a bounded lookahead/backup/parallel mechanism. There are some more difficult examples (involving lexical preferences) which appear to support the arc-ordering hypothesis. Sentences (185)-(186) are a typical minimal pair illustrating the difference between *see* and *know*, which cannot be distinguished in purely structural terms. Although we have not implemented a solution, we see no reason why these facts favor backup over lookahead (or parallelism).[103]

(185) I saw $[_s$ the horse raced past the barn].
(186) I knew $[_{np}$ the horse raced past the barn].

4.4 Pseudo-attachment

There are structurally ambiguous sentences, violating any well ordered set of principles; these should be recognized as ambiguous (or perhaps, vague). These present a problem for both Frazier's principles and our ordering heuristic. YAP detects the ambiguity with a marked rule. Frazier's two principles seem to conflict in this case. In the sentences below, minimal attachment would attach the *pp* high and late closure would seem to attach it low.

102. Martin (personal communication) has informed us that certain senses of *that* were more uniform in older forms of English. It is quite possible that we are missing a generalization in the various lexical forms of *that*.
103. In a parallel model, one could imagine that unusual lexical entries would take longer to fetch from memory, and hence, an unusual sense would lose the "race". In a lookahead system, it is possible to state the marked rules so they will trigger very rarely (only in the marked case).

(187) Sam hit the girl with a book.
(188) Sam hit [the girl] with a book. *high attachment*
(189) Sam hit [the girl with a book]. *low attachment*

There are several possible ways to deal with this apparent conflict.

(190) Define one of the two principles to avoid the problem. (Frazier's solution)[104]
(191) Cope with the possibility of conflict. (the "double finish" account)[105]
(192) Add an additional rule to cover the conflicting cases. (YAP's approach)

YAP has a marked rule to pseudo-attach (attach both ways)[106] when it sees both alternatives and decides that it cannot decide which is correct. This approach is completely consistent with Marcus' determinism hypothesis. YAP makes a single left to right pass over the input stream without backup. Once it pseudo-attaches, it will not retract the decision at a later date. In this way, Marcus' determinism hypothesis allows ambiguity, even though a deterministic PDA excludes ambiguity. The following sentences illustrate pseudo-attachment:[107]

(193) Put the block in the box on the table.
(194) He carried nothing to indicate that he was one of the group.
(195) We sighted the man with the binoculars.
(196) We never fought a bull with real courage.
(197) He hit the man with the stick.
(198) He seemed nice to her.

The estructure representation of these sentences in not a tree but rather a <u>directed</u> <u>acyclic</u> <u>graph</u> (DAG).[108] For example, $[_{pp-}$ to her] in (198) would have two mothers: the participial phrase $[_{vp-}$ seemed ...] and the adjectival phrase $[_{ap-}$ nice ...]. The multiple mothers should be interpreted as exclusive possibilities. This is a

104. Frazier [Frazier79 pp.143] argues that her late closure principle does not apply here because *the girl with a book* is a single package. As she defines late closure, it works on packages which are roughly six words long.
105. Suppose that the parser consisted of several parallel processes which were all competing against each other. The first process to finish would be the "winner" and its output would be taken as the preferred interpretation. When two processes finish at the same time, the sentence might be considered ambiguous/vague.
106. This idea was first suggested by Mitch Marcus. It is similar to Sager and Grishman's notion of *permanent predictable ambiguities*. [Grishman73]
107. Many of these sentences are from [Wales and Toner76].
108. A DAG is a general graph (of nodes and relations) with a condition excluding circular loops. Alternatively, a DAG is a generalization of tree where daughters may have multiple mothers.

convenient way to represent certain common structural ambiguities that occur in natural language. The cstructure of (198) would have the following representation:

(199) [$_{s}$ He [$_{vp}$ seemed [$_{ap-}$ nice PP$_{i}$] PP$_{i}$]]

where PP$_{i}$ = [$_{pp-}$ to her]

There are three interesting cases of pseudo-attachment illustrated by (200)-(202). In all three cases, down1 can attach to either up1 or up2. (See figure 8.) In (200), up2 optionally selects another daughter, whereas in (201) and (202), up2 obligatorily requires another daughter. In (201), unlike (202), there is another constituent, so pseudo-attachment is possible. There is a marked rule which considers the three possibilities.

(200) He [$_{up2}$ seems [$_{up1}$ nice [$_{down1}$ to her ...	*pseudo-attach*
(201) [$_{up2}$ Put [$_{up1}$ the block [$_{down1}$ in the box on the table ...	*pseudo-attach*
(202) [$_{up2}$ Put [$_{up1}$ the block [$_{down1}$ in the box.	*don't pseudo-attach*

Pseudo-attachment is not limited to just prepositional phrases; the YAP implementation generalizes the technique to work for any kind of xp- (pp-, ap-, or vp-), not just pp-. Consider the following examples:

Fig. 8. Pseudo-attachment

sentence: He seems nice to her.
input pointer:

[$_{s}$ he seems nice]
[$_{vp}$ seems nice]
[$_{ap-}$ nice]
= = WALL = =
[$_{pp-}$ to her]
[$_{pp}$ to her]
[$_{punct}$.]

The marked pseudo-attachment rule attaches down1 to both up1 and up2. YAP knows that it cannot disambiguate between the two possibilities.

(203) Put the block $[_{pp-}$ in the box on the table.
(204) I considered every candidate $[_{ap-}$ likely to win.
(205) He carried nothing $[_{vp-}$ to indicate that he was one of the group.

YAP uses a very similar technique to process certain well-known cases of ambiguous wh-movement[109] such as (206). These will be discussed when we consider wh-movement in chapter 8. (206) has two interpretations: (207) and (208). Both of these are represented within a single structure (209) where the trace NP-$_i$ has two mothers.

(206) Who(m) do you want to see?
(207) Who(m)$_i$ do you want to see t_i?
(208) Who(m)$_i$ do you want t_i to see?

(209) Who do you want NP-$_i$ to see NP-$_i$?

where NP-$_i$ = $[_{np-}]$

The pseudo-attachment technique follows a popular philosophy in artificial intelligence called delayed binding. The basic idea is to avoid making arbitrary decisions until there is enough information. This approach can be contrasted with an arc ordering technique (such as [Kaplan72]). In Kaplan's scheme, the possible decisions are ordered so the most plausible decisions are made first. In a delayed binding scheme, the system tries to avoid discriminating between possibilities as long as possible. In some cases, the system may never really distinguish between certain possibilities.[110]

---

109. *Wh-movement* refers to a class of constructions including relative clauses and wh-questions. These constructions relate a *wh-word* with a *gap* which is represented by a *t* (for trace). Traces are represented in YAP as phrases which dominate no words.

relative clause: I saw a boy *who*$_i$ you know t_i.
wh-question: *Who*$_i$ did you see t_i?

This will be discussed in more detail in chapter 8.
110. [VanLehn78] observed that informants sometimes claim they understand a sentence with multiple quantifiers until they are asked questions regarding quantifier scope. The subjects will often admit they hadn't considered the scope issue.

There are limitations to the particular implementation of delayed binding in YAP. It may be impossible to encode all grammatical ambiguous interpretations. We claim that pseudo-attachment can work for acceptable interpretations; the other grammatical interpretations are unacceptable. Unfortunately, it is very hard to test this claim.

It appears that pseudo-attachment cannot represent all CF interpretations because the device does not have CF generative capacity. One could view pseudo-attachment as annotating one of the attachments (the canonical attachment) with several alternatives. The weak generative capacity will be the same as the canonical structure; pseudo-attachment does not affect the <u>weak</u> generative capacity, only the <u>strong</u> capacity.[111] Assuming that YAP is equivalent to a deterministic PDA,[112] it has the weak generative capacity of a deterministic language (i.e. LR(k)). Since LR(k) languages do not include all CF languages, there are some CF languages which cannot be described using pseudo-attachment.[113] We claim that acceptable sentences can be described with pseudo-attachment.[114]

Pseudo-attachments should not be undone at a later date. There are certain problematic cases where the simple scheme described above will run into trouble. There are several possible replies. Some of the interpretations are probably unacceptable. Perhaps the rest could be processed with more lookahead. There are some problems with pseudo-attachment; nevertheless it is an interesting alternative to purely non-deterministic strategies.

111. The *weak* generative capacity is the set of <u>sentences</u> generated by a particular grammar. The *strong* capacity is the set of <u>derivations</u>. In general, the *strong* capacity is much larger since an ambiguous sentence corresponds to several elements in the strong generative capacity, but only one in the weak generative capacity. (Since the class of the machine (FS, CF, CS, TM) is tied to the weak generative capacity, pseudo-attachment can be implemented without moving to a higher computational class.)
112. It is conjectured that YAP would be a deterministic PDA if the stack bound were removed.
113. For example, there is no LR(k) grammar for an *inherently ambiguous* language.
114. This assumes that acceptable sentences form an LR(k) language. Even stronger, this result should follow from Marcus' Determinism Hypothesis and not from our FS hypothesis. (It trivially follows from the FS hypothesis since all FS languages are also LR(k).) Otherwise, it isn't clear how ambiguous parses could be found short of exploding the state space as suggested in chapter 1 when Marcus' Hypothesis was first mentioned. In other words, we are assuming that acceptable sentences (even with arbitrary center-embedding) are still weakly equivalent to an LR(k) language.

(210) Put the block in the box on the table PP* <u>into</u> <u>the</u> <u>basket</u>.
(211) I consider every candidate likely to seem XP-* <u>corrupt</u>.[115]

Sentences (210)-(211) illustrate a problem with pseudo-attachment; the final constituent, which is arbitrarily far from the decision point, selects the higher attachment as in (212)-(213). But without the final underlined constituent, the examples are highly ambiguous as (214)-(215) illustrate. The problem is that YAP has to look at the final constituent before it can determine whether or not to pseudo-attach. The final constituent might be arbitrarily far away.

(212) Put [the block in the box on the table PP*] [into the basket]. *unambiguous*
(213) I consider [every candidate likely to seem XP*] corrupt.

(214) Put [the block] [in the box on the table PP*]. *highly ambiguous*
Put [the block in the box on the table] PP*.
(215) I consider [every candidate] [likely to seem XP*].
I consider [every candidate likely to seem XP*].

We will make a simplifying assumption that the intermediate phrases all attach the same ways. Only the first and last few phrases in a sequence (XP*) can be pseudo-attached; it is assumed that the intermediate phrases all attach the same way. Consequently, the pseudo-attachment decision depends on just a few phrases (down1 and down3 of (216)), not on an unbounded number.

```
(216) [$_s$ put the block]
      [$_{vp-}$ put the block]
      [$_{np-}$ the block]
      ==WALL==
      [$_{pp-}$ in the box]                the first xp-
      PP*                                  the middle xp-*
      [$_{pp-}$ into the basket]            the last xp-
```

115. This example was suggested by Joan Bresnan.

Certainly there are numerous grammatical interpretations which cannot be described by this mechanism. For example, there are an unbounded number of grammatical interpretations; this mechanism only considers a bounded number:[116]

(217) I put [the block pp*] pp
(218) I put [the block pp_1] pp*
(219) I put [the block] pp*

We claim that the others are unacceptable (in the absence of positive evidence such as semantic bias). There could be marked rules to consider semantic or pragmatic clues.

4.5 Summary

The cstructure implementation has been outlined. Unless there is an applicable marked rule, the interpreter runs the phrase structure rules in an unmarked order. The unmarked order was chosen to be compatible with Frazier's two principles: late closure and minimal attachment. We have discussed several classes of marked exceptions (220)-(223). The description would be more attractive if the role of these marked exceptions could be minimized. This is an area for future research.

(220) early closure (the A-over-A closure principle)
(221) transformations
(222) non-minimal attachment
(223) pseudo-attachment

In the next chapter we will show how fstructure can be built from cstructure without violating memory and backup limitations.

116. There would be one other interpretation if *put* didn't subcategorize for an obligatory second object: *I saw [the block pp*].*

5. Functional Structure Implementation

The previous chapter sketched out YAP's basic machinery for constructing the constituent structure (cstructure), based solely upon category (n, v, np, vp, s, ...) information. The cstructure is an intermediate representation toward obtaining the predicate/argument relations (fstructure). Computing the fstructure involves a number of syntactic features (properties). It is easy to find minimal pairs such as (224)-(229) illustrating the necessity of certain syntactic features.

(224) That ball is round. *number*
(225) That balls are round is a fact.

(226) Have we eaten? *case*
(227) Have us eaten!

(228) Have the boys take the exam! *tense*
(229) Have the boys taken the exam?

Each node (phrase) has a number of syntactic features (eg. person, number, gender, case, tense and mood) and a number of grammatical roles (eg. subject, object, etc.) This chapter will outline a procedure for assigning features and roles. The problem is interesting because feature dependencies can cross seemingly unbounded distances. Nevertheless YAP has a procedure for manipulating features that doesn't violate the severe resource limitations (memory and backup). The feature manipulation problem is similar the inheritance problem [Fahlman77] [Martin79, 80], which is known to be very hard. Fortunately, the Bresnan-Kaplan linguistic theory provides us with just the necessary simplifying constraints.

Many parsers compile the feature information into the parts of speech (category), conflating constituent information (n, v, ...) with functional information. Perhaps the most extreme example is the Harvard Predictive Analyzer (HPA) [Kuno66] which used about 180 parts of speech to distinguish everything from number to subcategorization frames. We accept the proposal that the two structures should be independent.[117] In addition to her linguistic motivations, there are some computational advantages for dividing the problem in this way. It is often useful to delay certain decisions as long as possible. The HPA,

117. The independence property is central to the Bresnan-Kaplan framework though it has appeared in earlier models including ATNs.

with its 180 parts of speech, couldn't separate the distinctions which require immediate resolution from the ones that should be delayed. Consequently, it found many more ambiguities than most people consider. For example, the HPA finds three interpretations of (230) where most people notice only two, if that many. Some of these distinctions should be delayed (perhaps indefinitely). The multiple interpretations of *flying planes* are far more striking than the possible senses of *are*.

(230) They are flying planes.

(231) They are_{aux} $[_{\text{vp}}$ flying planes]
(232) They $\text{are}_{\text{copula}}$ $[_{\text{np}}$ flying planes]
(233) They $\text{are}_{\text{copula}}$ $[_{\text{vp}}$ flying planes]

YAP, as opposed to HPA, carries along multiple functional possibilities until there is some reliable information to resolve the various alternatives. In this way, YAP can manipulate feature dependencies over unbounded distances without violating Marcus' Determinism hypothesis.

5.1 Seemingly Unbounded Dependencies

We will illustrate a typical "unbounded" dependency in the features between two nodes and then show how the dependency can be captured with only finite memory. The method is in fact fairly general since it is based on the Bresnan-Kaplan linguistic theory.

(234) There is a problem.
(235) There are problems.

(236) *There are a problem.
(237) *There is problems.

There-insertion sentences such as (234)-(237) have two dependencies:

(238) subject-verb[118] agreement
(239) *there* agrees with its object

118. Grammatical roles (subject, object, predicate, etc.) will be undefined for the time being. The intuitive notions should suffice for the current discussion.

These dependencies can cross an unbounded amount of material as the following sentences illustrate:

(240) There seems likely to seem likely to seem likely ... to be a problem.
(241) There seem likely to seem likely to seem likely ... to be problems.

(242) *There seem likely to seem likely to seem likely ... to be a problem.
(243) *There seems likely to seem likely to seem likely ... to be problems.

In these raising[119] sentences, each embedded phrase takes an understood subject. The dependencies can now be stated locally, although they have unbounded consequences. That is, the highest subject agrees with the tensed verb and the most deeply embedded subject agrees with the object. Furthermore, all the understood subjects are related, so they inherit each other's constraints. Much of this chapter is concerned with the inheritance mechanism.

(244) There$_2$ seems x_4 likely x_6 to seem ... x_n to be a problem.

We will use a variable *x* as a place marker to represent the understood subjects. Now the two dependencies are local; *there*$_2$ agrees with *seems* and x_n agrees with *a problem*. Since the subjects are related, the procedure has unbounded consequences. Nevertheless the procedure does not require inordinate resources.

5.1.1 Grammatical Roles

The notion of subject is crucial to this formulation. The Bresnan-Kaplan analyses use a number of grammatical roles including subject, object, obj2 (second object), xcomp (adjectival, verbal, or prepositional complement), scomp (sentential complement) and predicate. Grammatical roles are assigned by structural and lexical constraints. For now, we will give an example to illustrate the intuitive notions:

(245) subj	I saw a boy.
(246) obj	I saw a boy.
(247) obj2	I gave a boy a ball
(248) xcomp	He seemed likely to be nice.
	He seemed to be nice.
	I gave a ball to a boy.

119. Raising is a particular linguistic construction which has received considerable attention in the linguistic literature (see [Postal74] for a long list of references).

(249) scomp It seemed that he was nice.
(250) pred I saw it.[120]

These are all slots in the fstructure. Grammatical relations are extremely useful for describing many linguistic phenomena (see [Bresnan80]).[121]

5.2 Constraint Propagation Solution

This feature manipulation procedure can be viewed as a constraint propagation problem [McAllester80] [Mackworth77] [Waltz75]. The problem is to propagate the agreement dependencies through the fstructure (a graph of grammatical roles). (See figure 9). Initially, all possible values are assigned; the number values are {singular, plural}.[122] Extraneous values are first weeded away by the lexicon and then by agreement constraints. In this way, multiple possibilities are carried along until there is sufficient information to disambiguate. YAP does not randomly try alternatives (non-deterministic); heuristic guessing is avoided whenever possible.

Figure 9 shows an fstructure after lexical specifications but before the constraint propagation. For example, the lexicon specifies that *a problem* is singular ({singular}) and *there* is either singular or plural ({singular, plural}). After propagating the two agreement constraints, x_2, x_4 and x_6 will all be singular (their number properties will be {singular}). The sentence, *There seem likely to be problems*, has a similar fstructure except x_2, x_4, x_6 and x_7 are plural instead of singular.

120. In the Bresnan-Kaplan framework, *pred* is a feature, not a grammatical role. We have placed it here because it is defined over a large set unlike the other features such as person, number and gender.

121. Chomsky (personal communication) has criticized grammatical relations as an inadequate explanatory theory. Although it is possible to describe the facts starting from grammatical relations, a truly explanatory theory would have to derive grammatical relations themselves. Chomsky argues that deriving grammatical relations from structural notions is the hardest part and consequently, the notion isn't very useful in a linguistic theory. This point is extremely controversial. Nevertheless, explanatory adequacy is somewhat orthogonal to processing issues; for our purposes "mere" descriptive adequacy is sufficient. (Descriptive adequacy is no simple task.)

122. We are assuming that features are defined over small sets of possible values. There are some theoretical difficulties associated with propagating grammatical roles since they have potentially unbounded ranges. The actual implementation has a special symbol (**undefined**) for the universal set of grammatical roles.

Fig. 9. Constraint Propagation
There seems likely to be a problem.
There$_2$ seems$_1$ x_4 likely$_3$ x_6 to be$_5$ a problem$_7$.

The fstructure graph (before propagating the agreement constraints) is given below (omitting certain details). The two agreement constraints are subject-verb agreement (x_2 with x_1) and there-insertion (x_6 with x_7). The constraints are sufficient to uniquely determine the number features ({singular}).

	before	after
x_1	pred: seems	
	tns: {pres}	
	subj: x_2	
	xcomp: x_3	
x_2	form: there	
	num: {singular, plural}	{singular}
x_3	pred: likely	
	subj: x_4	
	xcomp: x_5	
x_4	is-bound-to: x_2	
	num: {singular, plural}	{singular}
x_5	pred: there-be	
	subj: x_6	
	obj: x_7	
x_6	is-bound-to: x_4	
	num: {singular, plural}	{singular}
x_7	pred: a-problem	
	num: {singular}	{singular}

The two constraints are subject-verb agreement and there-insertion. In this framework, subject-verb agreement is enforced by intersecting the agreement features of a tensed node with its subject.[123] In figure 9, the number features of the tense node x_1 are intersected with its subject x_2, making x_2's number {singular}. Similarly, there-insertion constrains x_6 to agree with x_7, making x_6's number feature {singular}. By *is-bound-to* edges, the agreement constraints propagate all the way through the graph, making all the number features {singular}.

If the constraints were inconsistent, some slot would have no possible values, and the sentence should be ruled out. For example, the ungrammatical sentences, **There seem likely to be a problem* and **There seems likely to be problems*, are bad because their fstructures have no possible values (i.e. {}) for the number slots; one agreement constraint weeds out the value singular and the other removes plural. The ungrammatical sentences are functionally inconsistent.

If the constraints underdetermine the solution, some slots will have several possible values, and the sentence is considered vague (or perhaps ambiguous).[124] The number features in (251) and (252) are all {singular, plural} indicating a number ambiguity. In (251) there may be one or more "deer"; in (252), there is an ambiguity between the inner and the outer interpretation.[125] Sentence (253) has underdetermined tense ({pres, past}) since *put* is lexically ambiguous. The underdetermined cases illustrate that the evaluator can be so lazy it may never get around to making a decision.

(251) The deer might be nice.
(252) The family might be nice.
(253) I put it down.

123. Actually, tensed verbs don't have number features themselves, but rather assign number features to their subjects. For example, *seems* assigns singular features to its subject, although it is not singular itself. This point is important in examples like *That they seem to be nice is a fact* where the embedded clause is singular even though its main verb (*seem*) assigns plural features to its subject (*they*).
124. An ATN model can distinguish between *vagueness* and *ambiguity* because it has two mechanisms: underconstrained values (vague) and non-deterministic assignments (ambiguity). In our framework, we don't have the second mechanism and hence we cannot (currently) distinguish the two cases.
125. Collections can be viewed as many individual entities (inner), and hence plural, or they can be viewed as a single conglomerate (outer), and hence singular.

5.2.1 Representation Issues

Feature values are represented in bit vectors[126] so that each set (i.e. {singular, plural}) requires a constant amount of memory (independent of its size.) That is, the set {singular} and the set {singular, plural} require the same amount of memory. Unlike most non-deterministic systems, the ambiguity does not consume additional resources (time or space); the number feature requires exactly one bit vector in any case. These representation issues can have a fairly important impact on the overall performance of the system; it is often worthwhile to take advantage of the particular parallel construction of the machine at hand in order to avoid potentially expensive non-deterministic searching.

The features in figure 10 have been implemented.[127] Each *possible value* is represented by a single bit; 1 = possible, 0 = impossible. For example, if the *gen* and *dat* bits are set, then the case is either genitive or dative ({gen, dat}). In this representation, it is particularly easy to merge nodes; we simply intersect the two

Fig. 10. Features

feature	possible values
case	gen dat nom acc
gender	m n f
pnc	s1 s2 s3 p1 p2 p3
def	+ −
pro	+ −
tns	tnsless pres past +ing +en
mood	decl wh-q yes-no-q imperative exclamation subjunctive

126. A bit vector is an array of binary variables. It is very similar to standard set of binary valued features. We have chosen this representation for efficiency reasons: it requires minimum space and certain operations (store, fetch and merge) can be done in parallel because LISP has operations for performing logical operations in parallel on a single machine word (32 or 36 bits depending on the particular hardware).

127. Category (s, n, v, ...) is not implemented in this way because category features are not percolated through the fstructure like the others. For example, although there is good evidence that a noun phrase inherits a number value from its determiner (*this boy, these boys*), it is much harder to argue that it inherits a category value. Category is defined to be part of the cstructure.

bit vectors.[128] We are crucially depending on the fact that features range over a small finite set of possibilities.

5.2.2 No Disjunctive Constraints

There is a crucial linguistic assumption that enables the constraint propagation technique to work: there are no <u>disjunctive</u> constraints. It would not be possible to enforce a rule for example that required the first daughter to agree with <u>either</u> the second <u>or</u> third daughter. Disjunctive dependencies are known to be computationally difficult because they involve postulating several possible worlds which may have to be considered non-deterministically; fortunately they don't often appear in natural language syntax.[129]

---

128. Person and number have been combined (*pnc* = person/number code) because there are often disjunctive constraints between the two. For example, the noun *block* can take any person value and any number value, but the values are not independent (it cannot be *s3* = third person singular). This encoding trick is taken from Parsifal. Kaplan (personal communication) mentioned that his ATN parser used the same trick. (One could argue that *tns* and *pnc* are somewhat analogous; there are some words which have either *tns* features or *pnc* features, but not both. For example, the lexically ambiguous word *blocks* is either *pres* or *s3*, but not both. This idea has not been implemented.)

129. Martin (personal communication) knows of only one syntactic construction which suggests disjunctive dependencies. The partitive noun phrase *kind of dogs* might be either singular or plural. It seems to inherit its features from one or the other of its parts (but not necessarily both).

> What kind of dogs are those?
> What kind of dogs is the most popular?

Perhaps *kind* is not {singular}, but rather it is vague ({singular, plural}) between the inner and outer plural. The following pairs illustrate similar ambiguities.

> The bellows are coming apart.
> The bellows is being repaired.

> The committee are fighting among themselves.
> The committee is fighting the regulation.

This approach avoids disjunctive constraints, which are computationally problematic. Instead of postulating an arbitrary number of possible worlds, there is only one possible world which encodes the ambiguity (i.e. {singular, plural}). The system will not hypothesize which possibility is correct until there is sufficient information to be sure. In truly ambiguous sentences, the distinction will never be made.

> The deer might have done it.
> The fish shouldn't have.

This is consistent with the wait and see approach. (The set of possibilities (i.e. {singular, plural}) are stored in bit vectors; the information associated with a set it independent of the number of possibilities.)

Kaplan (personal communication) has suggested that lexical ambiguity and lexical redundancy rules are a very serious source of disjunctive constraints. His point is well taken, though progress is begin made. Robert Milne is currently working on the lexical ambiguity problem [Milne78a, 78b, 79, 80]. We will discuss our own solution to optional transformations (and lexical redundancy) in chapters 6-7.

5.2.3 Bind* is an Equivalence Relation

There is another useful simplifying assumption: the *is-bound-to* relation[130] forms natural equivalence classes.[131] We will replace the relation with its reflexive, symmetric, transitive closure: bind*. In figure 10, the embedded subjects are all bound to one another forming a single equivalence class (under bind*). Equivalence classes can be represented very efficiently; instead of storing each element individually, it is possible to store them collectively as a class, often saving considerable memory. The equivalence relation representation contains far less information than an arbitrary relation. This is very important for YAP, since there may be a bounded number of classes, even though there are an unbounded number of elements.

The equivalence property is a stipulation. We cannot currently explain why it fits the empirical data as well as it does. The theory would be more attractive if this assumption did not have to be stipulated.[132] It may be possible to explain it in terms of other independently motivated assumptions. Nevertheless, it seems to be consistent with the facts and it enables a computational optimization.[133]

YAP does not assign features to nodes individually, but rather to equivalence classes collectively. All the co-indexed subjects in figure 10 would share a single bag of features.[134] That is, x_2, x_4 and x_6 in figure 10 are represented collectively in the optimized fstructure (256) under x_2. In many parsers (including Marcus' Parsifal), each embedded subject would be represented individually.[135]

130. Our use of *is-bound-to* is very similar to transformational movements in Chomsky's framework. When we *bind* two positions, he would *move* a constituent from one position to the other.

131. This property is implicitly assumed in [Kaplan and Bresnan80].

132. There are some very interesting theoretical issues here. The Bresnan-Kaplan framework stipulates that binding is an equivalence relation; Chomsky makes no commitment either way. Which one is better? Suppose there were no empirical evidence to decide the matter. (Convincing evidence is very hard to come by.) On the one hand, the equivalence relation is an additional stipulation and hence it is undesirable. But on the other hand, the equivalence relation requires less information to represent (than a more general relation) and hence it is to be preferred. There is a certain advantage in having a more restrictive theory. It is not clear whether it is theoretically more desirable to have fewer stipulations or a more restrictive representation.

133. Although a processing argument alone is not adequate justification for adopting the proposed assumption (movement is equivalent to bind*), it should be sufficient motivation to study the proposal in greater detail.

134. Chomsky (personal communication) has proposed that case might be assigned to each index (i.e. each equivalence class), not to individual noun phrases. It is a fact that co-indexed noun phrases receive case exactly once.

135. This is inefficient in both space and time. In Parsifal, for example, it can take unbounded time to trace the binding pointers back to the lexical subject.

(254) There seems likely to be a problem.

(255) There$_2$ seems$_1$ x_4 likely$_3$ x_6 to be$_5$ a problem$_7$.

(256)	x_1	pred: seems
		tns: {pres}
		subj: x_2
		xcomp: x_3
	x_2, x_4, x_6	form: there
		num: {singular, plural}
	x_3	pred: likely
		subj: x_4
		xcomp: x_5
	x_5	pred: there-be
		subj: x_6
		obj: x_7
	x_7	pred: a-problem
		num: {singular}

Co-indexing is a unification procedure. Whenever two nodes are co-indexed, their features are merged (intersected) and placed in <u>shared</u> memory. Updating one node's features would affect the other because their features are being shared. In this way, an unbounded number of nodes could be affected with a single update, since they might all be sharing the same features. This is how the "unbounded" dependency in (254) can be realized with only limited working memory.

Although the dependency is "unbounded" in cstructure, it is bounded in fstructure, which uses the more efficient equivalence class representation. A grammatical role (i.e. subject) refers to an entire class (with potentially unbounded membership) such as $\{x_2, x_4, x_6\}$, not to an individual member. Consequently, it is possible for YAP to enforce these agreement constraints very efficiently in the fstructure since they mention only a bounded number of classes (grammatical roles).[136]

136. In the Bresnan-Kaplan framework, agreement dependencies are not allowed to reference more than four grammatical roles in a single rule.

Another attractive computational property of equivalence relations is *associativity*; they can be constructed in any order. x_2 could be unified with x_4 and then with x_6 or the other way around. The fstructure will turn out the same whether constraints are propagated cyclically[137] (bottom to top), inverse cyclically (top to bottom), or inside out. The results are invariant with the order of application. *Invariance is very convenient*; a parser can then enforce constraints in the most natural order (left to right).

Invariance does not follow from most definitions of movement because a lexical object cannot be moved until it has reached the source of the movement. Consequently it makes a difference whether movements are computed cyclically or not. Perhaps movement should be redefined to be associative.[138] Similarly, the ATN SENDR operation (which manipulates feature registers) is non-associative. This too could be redefined. Actually, part of the motivation for defining the Bresnan-Kaplan merge operator was to rid the asymmetry of the ATN SENDR [Kaplan (personal communication)].

5.3 The Bresnan-Kaplan Analysis of There-insertion

We will compare our analysis with the Bresnan-Kaplan analysis; YAP was designed so that it could easily incorporate many of their ideas. Consequently, we were able to borrow many analyses, such as the formulation of there-insertion, saving us considerable time and energy. We are not interested in reinventing all of linguistics; this thesis is mainly concerned with processing constraints.

The problem is to build a fstructure from the cstructure. The constraints on the fstructure come from the cstructure (e.g. the subject is the first *np* under tense) and the lexicon. A typical structural dependency relates a noun phrase in "subject position" (immediately dominated by a tensed clause) with the fstructure slot: *fsubj*. Similarly, there are lexical constraints indicating, for example, that *problem* is {singular}, *problems* is {plural} and *deer* is {singular, plural}. (257)-(258) link cstructure positions with grammatical roles; the remaining functional slots will be filled in by the lexicon.

137. [Freidin78] has observed that cyclicity is derivable from independently motivated assumptions. In this framework, the cyclic order generates the same results as any other order. We could interpret Freiden's results to say that order is irrelevant; the facts that were once explained using ordering constraints are covered under more general binding conditions.

138. A movement could for example leave a sink behind to swallow up the lexical phrase when it finally does arrive. There could be a well-formedness condition blocking final structures containing unfilled sinks. (This is similar to a free indexing scheme [Koster78].)

5.3.1 Structural Constraints

(257) up:s -> d1:np d2:vp
  d1 = subj(up)
  d2 = up

(258) up:vp -> d1:v (d2:np) (d3:xp-)[139]
  d1 = up
  d2 = obj(up)
  d3 = xcomp(up)

Examples (257)-(258) are a slightly modified form of Bresnan-Kaplan's notation. It has been changed to more closely resemble YAP's notation and to be easier to type.[140] Both (257) and (258) show a phrase structure rule followed by a number of constraint equations. For example, (257) gives an expansion for *s*; it has two daughters, the first is an *np* and the second is a *vp*. There are two constraint equations below the ps rule which fill in functional slots by a unification (co-index) operation. For example, the first equation, *d1 = subj(up)*, defines the *np* under *s* to be the subject,[141] by unifying the first daughter (an *np*) with the *subj* slot of *up* (an *s*). After the two nodes have been unified, they share the same memory so that further constraints on either node will affect the other. Hence the unification operator is the bind* equivalence relation; the classes are represented collectively in shared memory.

The second constraint equation *d2 = up* unifies the *head of a phrase* with its mother. This follows from x-bar theory [Jackendoff77] [Chomsky70] where phrases are defined as a projection of a head. For example, a noun phrase, such as the *the boy*, is a projection of its head noun *boy*. Similarly, an *s* is a projection of its head, a *vp*. Again, from x-bar theory, it follows that all features percolate up from the head. For example, the noun phrase *the boy* is singular because its head is singular. Similarly, $[_s$ I saw him] has past tense because its head *vp* has past tense. Functionally, one cannot distinguish a mother from its head, and consequently, they are

139. The pseudo-category *xp-* stands for one of the following: *ap-*, *vp-* or *pp-*.
140. YAP uses more mnemonic names: names like *d1*, *d2*, ... *dn* are replaced with *csubj*, *cobj*, *cxcomp*, ... The letter *c* indicates a cstructural relation, as opposed to an *f* for a functional role. Where we have used *up* and *d*, Bresnan and Kaplan would use up arrows and down arrows, respectively. Also, instead of numbering the daughters as we have, she writes the constraint equations underneath the appropriate daughter. Certain constraint equations can be understood as the unmarked case, so they need not be restated for each ps rule. See [Kaplan and Bresnan80].
141. Technically, the subject is the *fstructure* of the *np* under *s*, not the *np* itself. The subject does not include the *cstructure* of the *np* (category and surface daughters).

represented as a single unified node in fstructure.

5.3.2 Lexical Constraints

The remaining constraints come from the lexicon. A lexical entry looks very similar to a phrase structure rule. It defines a functional frame (instead of a constituent frame) with constraint equations between slots. We have used the dummy variables *a1, ..., an* instead of *d1, ..., dn* to distinguish functional arguments from constituent daughters. The following lexical entries are relevant to the example at hand:[142]

```
(259) seem -> a1:{vp-, ap-}[143]
      a1 = xcomp(up)
      subj(up) = subj(a1)

(260) likely -> a1:vp-
      a1 = xcomp(up)
      subj(up) = subj(a1)

(261) there-be -> a1:np-
      a1 = obj(up)
      num(subj(up)) = num(a1)
      form(subj(up)) = there
```

5.3.3 Well-Formedness Conditions

The functional structure is completely constrained by the constraint equations in the ps rules and the lexical entries.[144] The functional structure must meet three well-formedness conditions: completeness, coherence and consistency. Each lexical entry defines a functional frame where:

142. We will not discuss the internal structure of noun phrases at this time. For now, we will use the ad hoc predicate *a-problem* to represent the structure of $[_{np-}$ a problem$]$.

143. Technically, lexical predicates are not allowed to reference the cstructure (category and surface daughters). The Bresnan-Kaplan formulation replaces our *xcomp* with a *vcomp* (a *vp-* complement), a *acomp* (a *ap-* complement), a *pcomp* (a *pp-* complement) and a *ncomp* (a *np-* complement).

144. Subject-verb agreement was not described. There is a lexical entry for each form of the verb; each asserting a different constraint equation on the subject. For example, *seems* would have a rule like: *num(subj(up)) = {singular}*.

(262) each slot must be filled (completeness)
(263) and only those slots may be filled (coherence)
(264) and multiple assignments to a particular slot must be consistent

Sentences failing to meet these conditions are ungrammatical as (265)-(267) illustrate.

(265) *There is.	*incomplete*
(266) *It seems John to be a nice guy.	*incoherent*
(267) *There are a problem.	*inconsistent*

[Kaplan and Bresnan80] give an algorithm for instantiating lexical entries; we will not review it here since they were not concerned with the same resource limitations.

5.4 Implementation of Functional Structure

Examples (268) and (269) illustrate a typical phrase structure rule and a typical lexical predicate.[145]

```
(268) YAP's Notation                          Bresnan-Kaplan-like Notation

    (def-ps-rule finite-s s                   s -> d1:{s-, np-} d2:vp              ps rule
      (csubj obl (s- np-)                     d1 = subj(up)
        (action (merge down (get-fsubj up)))) d2 = up
      (chead obl (vp)
        (action (merge down up)))

    (def-pred seem-1 seem                     seem -> a1:{vp-, ap-}         lexical predicate
      (fxcomp obl (vp- ap-)                   a1 = xcomp(up)
        (action (subj-control up down))))     subj(up) = subj(a1)
```

YAP's ps-rules and lexical predicates share similar syntax, (269) and (270). Both of them are CF rules with Bresnan-Kaplan constraint equations encoded into the nonterminals (i.e. <term>). A <term> is defined as (271) below.

145. By convention, all functional slot names will begin with an *f*, whereas all constituent slot names will being with a *c*.

(269) (def-pred <predicate name> <stem> <term>*) *predicate rule*
(270) (def-ps-rule <ps-rule name> <category> <term>*) *ps rule*
(271) (<role> <OBLigatory, OPTional, or STAR> <possible categories> (action <lisp code>)) *term*

Recall that YAP's *attach* operation automatically advances the "dot" in a ps-rule pointer past a nonterminal. In addition to updating the ps-rule, advancing the "dot" also invokes the constraints associated with the nonterminal. That is, when YAP attaches a daughter to a mother, the daughter is given the <role> in the mother's frame, and secondly, the <action> field is evaluated with *up* and *down* bound to the mother and daughter, respectively.[146] For example, when YAP attaches down1 to up1 in (272), down1 becomes the *csubj* of up1 because the "dot" passes the *csubj* term. Furthermore, down1 becomes the *fsubj* of up1 because the *action* field specify that *down* (bound to *down1*) be merged with *up* (bound to *up1*).

(272) $[_S\ ]$ finite-s -> . csubj chead *before attaching*
fsubj: *empty*
csubj: *empty*

==WALL==
$[_{np\text{-}}$ I]
$[_v$ am]
$[_{det}$ det]

(273) $[_S$ I] finite-s -> csubj . chead *after attaching*
fsubj: $[_{np\text{-}}$ I]
csubj: $[_{np\text{-}}$ I]

$[_{np\text{-}}$ I]
==WALL==
$[_v$ am]
$[_{det}$ det]

146. The action field could contain an arbitrary LISP expression to be evaluated during an attachment, although by convention, the action fields merely update functional roles and syntactic features immediately connected to nodes in the buffers. It is *not* allowed to violate the FS hypothesis. (It would be an improvement to eliminate the *action* slot by classifying to possible *actions*.)

The fstructure parallels the cstructure in many ways. Just as we associated a ps pointer with every node, we will associate a predicate pointer with every predicate. When a daughter is attached to a predicate, the predicate pointer is advanced very much like the ps pointer is advanced. Advancing the pointer over a term invokes the relevant constraint equations. For example, attaching a *fxcomp* to *seems*, as in figure 11, invokes subject-control. That is, the daughter's understood subject is its mother's subject.

Fig. 11. PS Attach (revisited)

sentence: John seems to have left.
input pointer:

$[_s$ John seems$]$	finite-s -> csubj chead .
	seem-1 -> . fxcomp
	fsubj: $[_{np-}$ John$]$
$[_{vp}$ seems$]$	normal-vp -> chead . (cobj) (cxcomp)
	seem-1 -> . fxcomp
	fsubj: $[_{np-}$ John$]$
= = WALL = =	
$[_{vp-}$ to have left$]$	normal-vp- -> ccomp chead .
	have-1 -> fxcomp .
	fsubj: *empty*
$[_{punct}$.$]$	normal-x -> cword .

After attaching, up1's ps and pred pointers will advance invoking the constraint equations: *down1* becomes *up1*'s *cxcomp* and *fxcomp*, and *down1*'s *fsubj* is controlled by *up1*.

$[_s$ John seems$]$	finite-s -> csubj chead .	
	seem-1 -> fxcomp .	
	fsubj: $[_{np-}$ John$]$	
$[_{vp}$ seems$]$	normal-vp -> chead (cobj) (cxcomp) .	
	seem-1 -> fxcomp .	
	fsubj: $[_{np-}$ John$]$	
	fxcomp: $[_{vp-}$ to have left$]$	
	cxcomp: $[_{vp-}$ to have left$]$	
$[_{vp-}$ to have left$]$	normal-vp- -> ccomp chead .	
	have-1 -> fxcomp .	
	fsubj: $[_{np-}$ John$]$	*from subject control*
= = WALL = =		
$[_{punct}$.$]$	normal-x -> cword .	

For another example, there-insertion constraints are enforced when the *fobj* is attached, using the following lexical entry for the verb *to be*. When YAP attaches the *fobj*, it checks the *fsubj*; if it is the form *there*, YAP enforces number agreement, by merging the *num* feature of the subject and object.[147] This rule can have unbounded consequences since the fsubj can be passed down though an arbitrary number of raising verbals (like *seem* and *likely*).

```
(274) (def-pred be-1 be
          (fobj obl (np-)
            (action (if[148] (= *there (get-fsubj up)) (mergef (get-fsubj up) down num)))))
```

Bresnan-Kaplan's completeness, coherence, and consistency conditions are implemented using the predicate pointers. Completeness is a condition on closing; a node cannot close until all of its obligatory roles have been attached. Coherence is a condition on attaching; a daughter cannot attach unless it is an argument of its mother (or controlled by an argument of its mother).[149] Consistency is a condition on unification; inconsistent slots cannot be unified.

147. Note the difference between the *mergef* and *merge* functions. The former merges a particular feature (say *num*) whereas the latter merges all features. An equation like *up = down* merges all features whereas only the *num* feature is merged by an equation like *num(up) = num(down)*.
148. The lisp macro *if* is a simple conditional; it evaluates its second argument if the first argument returns true.
149. *Argument* is a linguistic notion which distinguishes positions selecting lexical items (*John, Mary, the table, ...*) from forms (*there, it, idiom chunks, ...*). The subject of *seem* is not an argument position because it can take forms as in (a). Lexical items which appear in that position are not arguments of *seem*, but rather of the *xcomp*. For example, in (b) *John* is an argument of *nice*, not *seem*.
(a) There seems to be a problem.
(b) John seems to be a nice guy.

5.5 An Example

The cstructure and fstructure for (275) are listed below. This example is very similar to Appendix 2 which traces the derivation more carefully.

(275) The boy was likely to sit?

```
(276) CSUBJ: [(NP-) the boy]                                   cstructure
          CHEAD: [(NP) the boy]
           CSPEC: [(DET) the]
           CHEAD: [(N) boy]
         CHEAD: [(VP) was likely to sit]
          CHEAD: [(V) was]
           CXCOMP: [(AP-) likely to sit]
            CHEAD: [(AP) likely to sit]
             CHEAD: [(A) likely]
              CXCOMP: [(VP-) to sit]
               CCOMP: [(COMP) to]
               CHEAD: [(VP) sit]
                CHEAD: [(V) sit]
```

```
(277) FSUBJ: [(NP-) the boy]                                   fstructure
          FSPEC: [(DET) the]
         FXCOMP: [(AP-) likely to sit]
          FSUBJ: [(NP-) the boy]
           FSPEC: [(DET) the]
          FXCOMP: [(VP-) to sit]
           FSUBJ: [(NP-) the boy]
            FSPEC: [(DET) the]
```

6. Lexical Transformations

The traditional arguments for complex models (e.g. TG and ATNs) suggest that simpler mechanisms (like YAP) cannot capture the full range of linguistic generalizations. This chapter will address this criticism.[150]

(278) "It is well known (cf. [Chomsky64]) that the strict context-free grammar model is not an adequate mechanism for characterizing the subtleties of natural languages. Many of the conditions which must be satisfied by well-formed English sentences require some degree of agreement between different parts of the sentence which may or may not be adjacent (indeed which may be separated by a theoretically unbounded number of intervening words). Context-sensitive grammars could take care of the weak generation of many of these constructions, but only at the cost of losing the linguistic significance of the 'phrase structure' assigned by the grammar (cf. [Postal64]). Moreover, the unaided context-free grammar model is unable to show the systematic relationship that exists between a declarative sentence and its corresponding question form, between an active sentence and its passive, etc."

There has always been some controversy over these arguments; currently, Gazdar [Gazdar79a,b,c] leads the opposition. The confusion stems from two very different interpretations of ***complexity***.

(279) <u>linguistic complexity</u>: the size of the grammar itself
(280) <u>computational complexity</u>: the time and space bounds for an ideal processor

In general, there is a trade-off between the two types of complexity; the size of a program (linguistic complexity) is typically inversely related to the power of the interpreter (computational complexity). Woods has adopted Chomsky's view that (279) should be optimized at the expense of (280).[151] Gazdar's position is

150. The following quotation is taken from [Woods70]. He is trying to justify augmenting his ATN model. An un-augmented ATN (a Recursive Transition Network RTN) has CF complexity.

151. Chomsky (personal communications) has said on many occasions that <u>weak</u> generative capacity (computational complexity) is completely irrelevant to the study of grammar. However, weak constraints can be used to limit the space of possible grammars. For example, if language (weak) is actually FS, then no strictly CF grammar (strong) can correctly describe the facts.

just the reverse.[152] Bresnan and Kaplan claim that it is possible to optimize both (to have your cake and eat it, so to speak). YAP was designed along these lines. It has very minimal computational complexity without sacrificing linguistic generalizations. This chapter will show how YAP captures many linguistic generalizations, greatly simplifying the grammar.[153] Chapters 6-9 discuss the following topics which are often used to "refute" a position like Gazdar's.

(281) Lexical Transformations	(passive, raising, there-inserion, ...)
(282) Local Structural Transformations	(aux-inversion, deletions, ...)
(283) Wh-movement	(wh-questions, relative clauses, ...)
(284) Conjunction	(vp deletion, gapping, elipses, ...)

This chapter will consider the following four constructions; other lexical rules are very similar.

(285) raising
(286) it-extraposition
(287) passive
(288) reanalysis

There is considerable controversy over these rules; we have adopted the lexicalist position which "compiles" the effect of these rules into the lexicon. That is, there are different lexical entries for *see* and *seen*; *see* is a transitive verb whereas *seen* is intransitive. Chomsky advocates a transformational position where passive and raising are subcases of *move-np*. Marcus has encoded Chomsky's analysis in a deterministic framework. This chapter will discuss a formulation of Bresnan-Kaplan lexical rules in YAP's framework.

152. It is widely believed that CF rules are inherently inadequate (in principle) to describe the facts. Gazdar (and others) give very good evidence to the contrary. It is theoretically possible to describe both active and passive sentences with two different CF rules. Similarly, it is possible to describe yes-no questions with yet another set of CF rules. Since there are only a finite number of transformations and only a finite number of base CF rules, one could apply all the transformations to the base, forming a large inelegant (but finite) set of CF rules which describe the facts. Gazdar's derivation could be viewed as a constructive "proof" that grammar has only CF (computational) complexity. (There are some apparently CS constructions to be considered: "respectively" in English, wh-movement in Swedish, subject-verb agreement in Dutch verbs, and Postal's Mohawk puzzle.)

153. Gazdar's system has meta-rules to achieve the same goals, though his solution tends to multiply the number of grammar rules by a rather substantial constant. Unfortunately, all known general CF parsing algorithms consume time proportional to the size of the grammar, and hence Gazdar's solution will slow down parsing time by a rather substantial constant.

6.1 The Lexical/Transformational Debate

The last chapter demonstrated a lexical formulation of there-insertion (coupled with raising). The understood subjects were related to each other in the fstructure by lexical constraint equations. Chomsky would achieve a similar result by representing the understood subjects as traces (empty noun phrases) in the cstructure. Instead of using lexical constraint equations to bind the traces, he uses a syntactic transformation called *move-np*.

The differences between these two positions are very subtle. We will review one argument for each side to illustrate the flavor of the debate. Neither of these arguments is definitive; there is a large literature of replies and counter-replies. The arguments should demonstrate that competence issues (lexical versus transformational) are orthogonal to performance. The state of performance models is not sufficiently sophisticated to distinguish subtle competence issues. It is doubtful whether performance models can ever distinguish certain matters of competence.[154] Both the lexical and transformational positions are internally consistent (for the most part) and equally parsable (Marcus used a transformational approach). We chose the lexical position for its very attractive representation of features (described in the last chapter). Although it may be possible to devise a similar scheme in a transformational framework, the lexical representation was available when YAP was being designed. The debate has concentrated on two points:

(289) Do move-np rules (passive, there-insertion, raising, etc.) leave a trace?

John was seen.	*lexical*
John_i was seen t_i.	*transformational*

(290) Do infinitives take lexical subjects?

I believe $[_{np\text{-}}$ John$]$ $[_{vp\text{-}}$ to be a nice guy$]$	*lexical*
I believe $[_s$ John to be a nice guy$]$	*transformational*

154. An extreme functionalist position might suggest that all competence issues are ultimately specified by processing considerations. This seems most unlikely.

The following two arguments debate point (289).

6.1.1 The Wanna Argument

The Wanna argument [Bresnan78] demonstrates that there-insertion "must" be a lexical rule since it does not leave a trace (an empty noun phrase in cstructure). In English, certain verbals (e.g. *want*, *going*) can optionally contract with the word *to* as in (291) and (292).

(291) I want to go home.
I wanna go home.

(292) I'm going to go home.
I'm gonna go home.

Want + *to* cannot contract over a trace. Hence contraction is blocked in (293) by the trace of wh-movement, but permitted in (294) where the trace does not intervene.

(293) Who$_i$ do you want t$_i$ to see Bill?
*Who do you wanna see Bill?

(294) Who$_i$ do you want to see t$_i$?
Who$_i$ do you wanna see t$_i$?

The question is: does move-np leave a trace? Is there-insertion a lexical rule as in (295) or a transformation as in (296)? If there-insertion leaves a trace, then contraction should be blocked as in wh-movement. But contraction is permitted, so there-insertion "cannot" leave a trace.

(295) There is going to be a movie about us. *lexical*
(296) There$_i$ is going t$_i$ to be a movie about us. *transformational*

(297) There's gonna be a movie about us.

6.1.2 The Away Argument

[Williams80] argues that the durative particle *away* occurs only with intransitive verbs as demonstrated by (298)-(301).

(298) The dial is spinning away.
(299) *John is spinning the dial away. (wrong meaning)
(300) John is hitting away at Bill.
(301) *John is hitting Bill away.

He then observes that *away* can occur with lexically derived intransitives (where there is no trace), but not with syntactically derived intransitives (where there is a trace).

(302) John is eating away. *lexically derived*
(303) *Who$_i$ is Bill hitting t$_i$ away. *syntactically derived*

If passive is a lexical rule, then it should allow *away* by analogy with (302); if it is syntactic (leaving a trace), it should block *away* as in (303). In fact, *away* cannot occur with passives, so move-np "must" leave a trace.

(304) *Bill$_i$ was being hit t$_i$ away by Fred.

Neither position is conclusive. Having adopted the lexicalist position, we should show how linguistic generalizations can be encoded within the lexicalist framework. Furthermore, the encoding is subject to the processing limitations (finite state and determinism).

6.2 Raising

The last chapter illustrated a lexical analysis of raising; we will summarize the analysis here. There are two types of raising rules: raising-to-subject (305) and raising-to-object (306). In both cases, there is a raising verbal in the higher matrix (e.g. *seem, promise, likely, persuade*) which determines the type of raising. In the *seem* case (raising-to-subject), the embedded subject is bound to the higher subject; in the *persuade* case (raising-to-object), the embedded subject is bound to the higher object. Bresnan-Kaplan constraint equations elegantly capture both cases.[155]

155. The term *raising* comes from the old analysis where transformations literally raised the embedded subject up to the higher matrix. See [Postal74] for a defense of the traditional analysis.

(305) subj(up) = subj(xcomp(up)) *raising-to-subject*

John seems to be a nice guy.
John promised Mike to be a nice guy.
John is likely to be a nice guy.
John struck Mike as likely to be a nice guy.

(306) obj(up) = subj(xcomp(up)) *raising-to-object*

John persuaded Mike to be a nice guy.
John forced Mike to be a nice guy.
John convinced Mike to be a nice guy.

6.3 Auxiliaries

YAP analyzes auxiliaries as raising-to-subject verbs; they all select a verbal fxcomp and subject control. Unlike raising verbs, auxiliaries select participial *tns*[156] features whereas raising verbals generally select infinitival *tns* features.

(307) I <u>was</u> [$_{\text{xcomp}}$ going]. *auxiliaries*
I <u>will</u> [$_{\text{xcomp}}$ go].
I <u>have</u> [$_{\text{xcomp}}$ gone].

(308) I <u>seem</u> [$_{\text{xcomp}}$ to go]. *raising*

Modals (*can, may, will, ...*) and *do* select *tnsless* complements, *have* takes +*en*, and *be* assigns either +*ing* or +*en*.[157] For example, the predicate for *be* would look something like:

156. The *tns* feature takes either tense or participle values (since the two have complementary distributions.) The possible values are: pres, past, tnsless, +en and +ing.
157. Many analyses separate the two forms of *be* into an active and a passive entry. Our formulation is more consistent with the wait and see philosophy. We claim there is only one copula *be* which selects an *xcomp* marked with either active or passive inflection ({+*ing*, +*en*}). The active and passive interpretations are determined by the participle's predicate, not by the copula.

```
(309) aux-be -> a1:vp-
      a1 = xcomp(up)
      tns(a1) = {+ing, +en}
      subj(a1) = subj(up)
```

Auxiliaries can nest freely to form sentences like the following:

(310) I would have been taken.
(311) I would have been taking the ball.

There are a few constraints which limit the possibilities. Modals and *do* have no participial forms (in their auxiliary senses)[158] so they must appear in positions requiring present or past inflection. In other words, they must be directly dominated by a tensed clause because that is the only tensed position. For example, (312) is ungrammatical because *will* does not have a *tnsless* form which would normally be required after *would*. (313) is out for similar reasons.

(312) *I would will have ...
(313) *I would do have ...

Even with these constraints, the raising analysis seriously over-generates. One could fix this problem using a small set of motivated features as in [Akamajian79]. Currently, YAP will accept sentences like (314). It is possible that these should be excluded on semantic or pragmatic grounds like (315) which are syntactically

158. Certain modals are easily mistaken with main verb forms, which have very different morphology and distributions.

I should *can* you for that.
I *had* the boys take the exam.
I *did* it.

It isn't clear how a parser can distinguish the two forms. YAP has some marked rules to disambiguate a few cases. Lexical ambiguity is a very hard problem.

well-formed, though semantically questionable.[159]

(314) *I have been having been having ...
*I have had had ...

(315) ?It seemed to seem to seem ...
?It is likely to be likely ...

Except for this problem, the raising analysis is extremely simple and efficient. See [Akmajian79] for a critical review of these proposals and some alternatives.

6.4 It-extraposition

The raising analysis has a number of manifestations; it has played a crucial role in there-insertion and auxiliaries. It also turns out to be important in it-extraposition, illustrated by (316)-(318) below.

(316) <u>It</u> was believed <u>that</u> <u>I</u> <u>would</u> <u>go</u>.
(317) <u>It</u> was promised <u>that</u> <u>I</u> <u>would</u> <u>go</u>.
(318) <u>It</u> seemed likely <u>that</u> <u>I</u> <u>would</u> <u>go</u>.

It-extraposition is similar to there-insertion; both cases illustrate a dependency between a subject and a deeply embedded constituent. In there-insertion, the "dummy" form *there* depends upon a deeply embedded <u>noun phrase</u> such as *a problem*; in it-extraposition, the "dummy" *it* depends upon a deeply embedded <u>clause</u>.

---

159. We could suggest some more filters to exclude some of the additional cases. For example, *Have* doesn't take +*ing* in its auxiliary form.

I have taken it.
*I was having taken it.

A second condition blocks two adjacent verbs with +*ing* inflection.

*[... +*ing* +*ing* ...]
*I am being being ...

These filters are merely descriptive; a true theory would explain these facts.

(319) There seemed likely to seem likely ... to be a problem.
(320) It seemed likely to seem likely ... that I would go.

YAP uses a similar mechanism in both cases: just as there are lexical entries which check their fsubj slot for the form *there*, there are lexical entries which check for *it*. Since subjects can be raised arbitrarily far, it-extraposition can have unbounded consequences.[160]

```
(321) (def-pred be-1 be
        (fobj obl (np-)
         (action (if (= *there (get-fsubj up)) (mergef (get-fsubj up) down num)))))

(322) (def-pred likely-1 likely
        (fscomp obl (s-)
         (action (if (= *it (get-fsubj up)) (merge (get-fsubj up) down)))))
```

The form *it* in (323) is co-indexed with the scomp (sentential complement) to distinguish it from the pronominal *it* in (324). The two interpretations have different semantics.

(323) It seemed that we were nice. (meaningless *it*)
(324) It seemed to be nice. (pronominal *it*)

Similar comments apply to *there*; (325)-(326) demonstrate the different semantics of *there*.

(325) There was a problem. (meaningless *there*)
(326) I went there. (pronomial *there*)

6.5 Passive

Our *passive* analysis depends on the formulation of auxiliaries as raising verbs. Passive participles do not stipulate the auxiliary. It happens that *to be* is the only auxiliary that can take a passive participle.[161] This is

160. Note that it-extraposition merges every feature associated with the subject whereas there-insertion only merges the *num* feature. Hence, it-extraposition uses the *merge* function whereas there-insertion uses the *mergef* function.
161. Except for *have*, all other auxiliaries block *+en* participles. (They merge some other ms feature with their fxcomp.) For some unexplained reason, *have* blocks passive interpretation of its fxcomp.

purely accidental; passive participles are found in many other constructions without the verb *to be.*[162] The verb *to be* is identical in both (327) and (328); the difference is restricted to the participial phrases *seeing me* and *seen.*

(327) John was seeing me.
(328) John was seen.

There are two lexical entries, one for *seeing* (329) and one for *seen* (330), which are related by a lexical redundancy rule to capture the passive generalization.

```
(329) active-see -> a1:np- a2:np-
        a1 = subj(up)
        a2 = obj(up)

(330) passive-see -> a1:np-
        a1 = subj(up)
        tns(up) = +en
```

In the Bresnan-Kaplan framework, all lexical entries are "tried" non-deterministically; structures meeting the functional well-formedness conditions (coherence, completeness, and consistency) are considered valid interpretations. This is a perfectly reasonable competence model; however, it may have two problems as a model of performance:

(331) very large lexicon
(332) non-determinism

162. Here are three constructions involving passive participles:

a <u>fallen</u> leaf
He seemed <u>persuaded</u> to leave.
I saw a horse <u>taken</u> past the barn.

There is a considerable literature discussing passive generalizations; our formulation is consistent with the lexical analyses, although many of the details have not been implemented.

YAP uses a *virtual* lexicon to alleviate problem (331). Instead of storing all the lexical entries literally in a huge array, YAP stores only the core entries; other entries are generated upon demand. Viewing the lexicon as a black box, it shouldn't be possible to distinguish the real entries from the virtual ones. The virtual lexicon is very analogous to virtual memory systems which page address locations into real memory upon demand. These schemes take advantage of a space/time trade-off.[163]

Determinism is more difficult to arrange. How can YAP decide which lexical entry to use? The lexical ambiguity problem is extremely difficult. In this case, there are some fairly good heuristics. The unmarked case is triggered by a +*en* morphological feature, though there are several marked rules to disambiguate some of the more difficult cases. These rules may seem ad hoc, but they do have to be stated in one way or another. Perhaps we will find an explanation someday; for now, we will make do with a descriptive theory.

(333) John was seen.	(the unmarked case)
(334) John has seen Bill.	(perfect construction)
(335) The horse raced past the barn. The horse raced past the barn fell.	(+*en*/+*ed* ambiguity)

There are two exceptional cases: the perfect construction (333) and the +*en*/+*ed* morphological ambiguity. The perfect construction blocks the passive rule from applying to its complement. This fact is stated in the lexical entry for *have*. The morphological problem in (335) is disambiguated by the unification procedure. The two senses of *raced* ({+en, past}) are merged (intersected) with the two senses of a tensed clause ({pres, past}) producing a unique result (see figure 12).

YAP has a production rule to generate a passive predicate pointer when it is needed. It looks something like the following, although a number of details have been omitted for clarity.[164]

---

163. Page faults (generating lexical entries on the fly) become less and less probable as more and more lexical entries are added to the core lexicon. It may be more efficient to include redundant information in the lexicon which is frequently accessed, thus reducing the chance of a page fault. In other words, it may be worthwhile to sacrifice some linguistic complexity to achieve improved computational complexity.

164. For example, there has to be a mechanism to prevent the rule from re-applying arbitrarily often to the same predicate. There is an uninteresting lisp expression in the pattern to accomplish this.

Fig. 12. Disambiguating +en/+ed
sentence: The horse raced past the ...

[$_s$ the horse]	tns: {pres, past}
==WALL==	
[$_{vp}$ raced]	tns: {past, +en}
[$_p$ past]	
[$_{det}$ the]	

There is a constraint equation which unifies a clause with its head (the vp). When the head is attached the constraint equation is evaluated, disambiguating the <u>tns</u> features. The two senses of *raced* ({+en, past}) are merged (intersected) with the two senses of up1 ({pres, past}) producing a unique result.

[$_s$ the horse raced]	tns: {past}
[$_{vp}$ raced]	tns: {past}
==WALL==	
[$_p$ past]	
[$_{det}$ the]	

```
(336) (defrule passive trans
          (pattern () (= +en))
          (action (passivize-pred down1)))
```

The function <u>passivize-pred</u> transforms down1's active predicate pointer into a passive one. (It simply replaces the fsubj slot with the fobj slot.)[165] This should have the same external appearance as though there were passive predicates stored in the lexicon. It is merely a space/time trade-off.

6.6 Reanalysis

In general, prepositional objects do not passivize. For example:

165. Unfortunately, this does require copying the predicate pointer.

(337) *The ball was gone to.
*The river was seen at.
*The boy was taken the ball from.

However, there are some marked cases where passive is possible. To account for these facts, it has been proposed that certain verb-particle combinations (e.g. *arrive at* and *look at*) can reanalyze into a single verb complex. The reanalyzed form (338) can passivize, unlike (339), because *the solution* is a verbal object whereas *the station* is prepositional object.

(338) They [$_v$ arrived at] [$_{np-}$ the solution].
The solution was arrived at.

(339) They arrived [$_{pp-}$ at the station].
*The station was arrived at.

Since YAP is not capable of distinguishing the semantic difference between *the solution* and *the station*, it cannot distinguish (338) from (339). When syntactic clues are sufficient as in (340)-(341), YAP correctly performs the reanalysis.

(340) I looked at the picture.
The picture was looked at.

(341) I went to the ball.
*The ball was gone to.

The difference between *look* and *go* is stated in the lexicon; *look* reanalyzes with *at*, but *go* does not reanalyze with *to*. The lexical entry for *look at* is listed below. Notice that it takes a direct object, not a prepositional object.

```
(342) (def-pred look-at-1 look
        (fsubj obl (np-))
        (fcase obl (p)
        (fobj obl (np-)))
```

We have seen how a number of lexical rules (raising, it-extraposition, there-insertion, auxiliary formation, passive, and reanalysis) are formulated in YAP. This shows that many of the generalizations can be captured by a relatively simple device.

7. Local Structural Transformations

The last chapter demonstrated several rules which operate on predicate pointers (fstructure). This chapter will discuss structural transformations which operate on constituent structure (cstructure). There are some important differences between lexical and structural rules.

(343) Lexical rules are local in *fstructure*; structural rules are local in *cstructure.*
(344) Structural rules have no lexically marked exceptions.
(345) Lexical rules are structure preserving.[166]

By these criteria (which are admittedly very pro-lexicalist), it is very hard to find suitable candidates for a structural rule. (343) is not very discriminating; as we have seen, it is generally possible to state many rules in either the fstructure or the cstructure. (344) is very pro-lexicalist, since almost every linguistic generalization has an exception. Only (345) establishes a class of structural rules; some rules (e.g. root transformations) are not structure preserving.[167] This section will analyze two root transformations: <u>aux-inversion</u> and <u>imperative</u>.

The structure preserving property [Emonds76] is analogous to side-effect[168] free (applicative) programming; both moves attempt to establish an invariant representation which remains intact after an arbitrary number of transformations (function calls). Linguists have found the invariance notion to be useful for describing grammar; computer scientists have discovered invariance important in program verification. It is generally agreed in both fields that structure preserving (applicative) formulations are desirable.

166. [Emonds76] postulates that transformations divide into two categories: *Structure-Preserving Transformations* and *Root Transformations*. The former introduce or substitute a constituent C into a position in a phrase marker held by a node C; root transformations move, copy and insert a constituent in root clauses.
167. Actually the case is not so clear; there may be ways to reformulate these transformations to be structure preserving. For example, [Kaplan and Bresnan 80] present a structure preserving analysis of *imperative.*
168. A program is said to cause *side-effects* if it modifies data structures in a non-invertible fashion. In general, it is possible to avoid side-effects; there is a school of computer scientists who advocate completely side-effect free programming. This position is somewhat analogous to the lexicalist school of linguists who advocate side-effect free analyses.

7.1 Aux-inversion

Perhaps the best example of a structural transformation is the so-called aux-inversion rule which has applied to (346)-(350).[169]

(346) Have I taken the ball?
(347) Which balls have I taken?
(348) Never have I taken so many balls!
(349) Under no circumstances am I permitted to release these documents.
(350) Nowhere could he find an alpaca carpet.

YAP's aux-inversion rule undoes the inversion by switching the buffer cells containing the auxiliary and the subject noun phrase, thus capturing the linguistic generalization without increasing the computational complexity (memory is still severely bounded). The aux-inversion rule inverts down1 and down2 as illustrated in (351). It also labels up1 with the mood feature {wh-q, yes-no-q} to distinguish the sentence from its declarative form.[170]

(351) sentence: Have I taken the ball?
input pointer: the ball?

before	after
$[_s]$	$[_s]$
= = WALL = =	= = WALL = =
$[_v$ have$]$	$[_{np}$- I$]$
$[_{np}$- I$]$	$[_v$ have$]$
$[_v$ taken$]$	$[_v$ taken$]$

A simple form of the aux-inversion rule is shown below.[171]

169. Only *yes-no* and *wh-questions* have been implemented; the other cases shouldn't be too much more difficult.
170. This doesn't work in the preposed adverbial case. *Never have I seen so many balls!* Bob Berwick (personal communications) has suggested that the inverted forms share a common LF (logical form) interpretation which distinguishes them from declarative sentences.
171. The last term of the pattern could be an arbitrary lisp predicate which must be true in order for the rule to match. In practice, the predicates tend to test features of nodes in the buffer. In this case, the predicate *crole-can-advance?* is testing if up1 is looking for a subject. Some details have been suppressed for clarity. For example, there are some agreement constraints which will be discussed later in this chapter.

```
(352) (defrule aux-inversion trans
          (pattern (=root) (=auxverb =np-) (crole-can-advance? up1 'csubj))
          (action (invert) (setfeat up1 (yes-no-q wh-q) mood)))
```

Aux-inversion is possible when up1 contains a root clause[172] looking for a subject, and the lower buffer holds the inverted auxiliary/np- pattern.[173] This rule was taken almost directly from Marcus' Parsifal.

7.2 Imperative

Imperative is a deletion rule which applies to root clauses.[174] The parser simply restores the deleted elements and finishes the sentence as if nothing had been missing. Given a sentence like (353), YAP will insert the words *you will* into the lower buffer, undoing the imperative transformation. YAP will finish the sentence as if it had been parsing (354). As in aux-inversion, the transformation adds a mood feature to distinguishes the transformed sentence (353) from the untransformed sentence (354). The rule is given as (356) below.[175]

---

172. The highest clause is a root clause. There are some other instances of root phenomena which YAP does not currently handle. For example, *I said, "what are we going to do?"*

173. The following verbs act as auxiliaries in English: *be, have, do, can, will, may, shall, must,* and perhaps a few others. There is another marked rule (described in the next section) which blocks aux-inversion when *do* and *have* (in American English) are being used in their mainverb senses as below:

Have the boys take the exams! (mainverb)
Who had the boys take the exams?
Do it!
Who did it?

Have the boys taken the exams? (auxverb)
What have the boys taken?
Did it bother you?
Who does it bother?

It is an unexplained fact that *be* and the British use of *have* invert (even in the mainverb sense).

174. [Kaplan and Bresnan80] give a lexical analysis of imperative.

175. This rule was also taken from Marcus' Parsifal. There is one difference: his rule drops the word *you* into the buffer, not the words *you will*. YAP will parse (a) like (b); Parsifal will parse it like (b).

(a) Be good!
(b) You will be good!
(c) *You be good! (wrong meaning)

YAP drops the *will* to absorb the tense constraint on root clauses; root clauses are tensed, except for imperatives which have no overt tense marker.

(353) Take the ball!
(354) You will take the ball!

(355) <u>before</u>	<u>after</u>
$[_s]$	$[_s]$
==WALL==	==WALL==
$[_v$ take$]$	$[_{np}$- you$]$
$[_{det}$ the$]$	$[_v$ will$]$
$[_n$ ball$]$	$[_v$ take$]$
	$[_{det}$ the$]$
	$[_n$ ball$]$

```
(356) (defrule imperative trans
        (pattern (=s) (=v) (and (=tnsless[176] down1) (crole-can-advance? up1 'csubj)))
        (action (setfeat up1 imperative mood) (drop-words you will)))
```

7.3 Differential Diagnosis

It happens that both <u>aux-inversion</u> and <u>imperative</u> have very similar patterns. In examples like (357)-(359), there is some difficulty deciding which transformation should apply. Some cases, such as (359), are grammatically ambiguous, and hence, it is not possible to disambiguate using just the rules of grammar (competence).[177]

(357) Have the boys take the ball! *imperative*
(358) Have the boys taken the ball? *inversion*

(359) Have the eggs fried ... *ambiguous?*

A non-deterministic system could "try" both rules, accepting all analyses that happen to work out. A deterministic system is posed with a difficult problem; both transformations (aux-inversion and imperative) cause side effects which cannot be undone. A deterministic machine has to make the right decision the first time; there will be no recovering if it selects the wrong transformation. This section will discuss procedures

176. The predicate *=tnsless* tests for null inflection.
177. The ambiguity may not be realized in performance. Marcus claims there is a strong preference for inversion in the unmarked case, though the marked interpretation can be forced by semantic and pragmatic biases.

for deciding which transformation should apply.

Marcus believes this problem results from a lexical ambiguity between the two senses of ***have***.[178] The auxiliary *have* undergoes inversion as in (360) unlike the main verb *have* (in American English). Hence, if we could distinguish the two forms of *have*, we could decide which transformation should apply. Marcus invokes a marked rule (360), called <u>Have-diag</u>, to disambiguate differentially[179] between the two senses of *have*.

```
(360) pattern:
          down1: [$_V$ have]
          down2: [$_{np-}$ <any> ]
          down3: [$_{<any>}$ <any> ]

      If down3 is tnsless or down2 is plural (first or second person),    marked exception
          then run imperative next.[180]
      Otherwise, run aux-inversion next.                                  unmarked default
```

The default path (inversion) is taken, unless there is marked evidence to the contrary. Marcus claims that the marked information must appear *in the next three constituents*. He has some empirical evidence indicating that many people cannot disambiguate (361)-(362) because there is no disambiguating information within the specified lookahead. In (363)-(364), the default interpretation (inversion) is blocked locally by the underlined words, and hence, (363)-(364) receive the exceptional interpretation (imperative).

178. The main verb sense of *have* inverts more freely in British English.
American: Do you have a match?
British: Have you a match?

179. The term *differential diagnosis* was derived from medical applications. It is believed that doctors have precompiled rules to differentiate between medical conditions which have similar symptoms but require very different diagnoses. [Davis77] refers to these rules as *meta-rules* because they reason about rules. This is a very powerful technique, though potentially expensive.

180. Actually, this rule has a slight flaw: it fails to distinguish *Have <u>I</u> eaten?* from *Have <u>me</u> eaten!*. This suggests that *case* features (in addition to person/number) should be used to disambiguate. Neither YAP nor Parsifal use *reflexive* features to disambiguate. For example, compare:
Have <u>yourself</u> completely taken advantage of, for all I care!
Have <u>you</u> completely taken advantage of every chance?

(361) [$_1$ Have] [$_2$ the packages] [$_3$ delivered] tomorrow. *unmarked*
(362) [$_1$ Have] [$_2$ the soldiers] [$_3$ given] their medals by their sweethearts.

(363) Have <u>them</u> delivered tomorrow. *marked*
(364) Have the soldiers <u>take</u> their sweethearts to the dance.

This approach works in a large number of cases. Like other marked rules, it suggests three important questions:

(365) How are diagnostics restricted?
(366) Is there any empirical support for this approach?
(367) How many diagnostics will be needed?

Marcus' lookahead buffer addresses question (365). The three constituent limit is consistent with the empirical evidence mentioned above (361)-(364) and the garden path phenomena.[181] Although Marcus' approach has these desirable characteristics, there is some concern that a complete grammar would require too many diagnostics. Diagnostics are used when there is a lexical ambiguity that would lead to multiple cstructures. The number of diagnostics becomes troublesome when they compare two or more transformations at a time, and hence, there may be a combinatoric number of diagnostics. It is quite reasonable to place conditions on a transformation one at a time; the problem comes when multiple transformations must be compared *differentially*. It is possible that differential diagnosis may require an inordinate number of rules. We will reformulate Marcus' Have-diag as follows:[182]

(368) Aux-inversion is blocked when any of the following conditions cannot be met: *competence*
down1 has *pres* or *past* inflection
down1 can take down2 as subj (agree in person, number, gender and case)
down1 can take down3 as xcomp (agree in inflection)

(369) Imperative is blocked when aux-inversion can apply. *performance*

181. Like other performance limitations, the buffer length is subject to a certain amount of individual variation.
182. We accept Marcus' assumption that no rule can access beyond down3, although additionally, we allow rules to access up1, up2 and up3. This is a performance limitation on backup/lookahead. It seems to be subject to the same idiosyncratic behavior that plague other performance constraints (e.g. individual variation).

Our formulation has three advantages over Marcus':

(370) Clear separation of competence and performance
(371) Covers a wider range of cases
(372) Fewer differential rules

It is important to separate competence and performance; performance filters such as (369) are generally more idiosyncratic than statements of competence (368). Performance phenomena are often subject to semantic and pragmatic biases, garden path behavior and variation from one informant to another. For example, (369) is subject to a certain amount of individual variation as Marcus has observed; it is unlikely that (368) can be overruled in the same way.

Our statement is more general than Marcus'; His rule only applies to *have*; our formulation covers all auxiliaries, including *did* and *was* as illustrated in (373)-(376).

(373) <u>Who</u> <u>did</u> it? *no inversion*
(374) Who <u>did</u> <u>it</u> bother? *inversion*

(375) <u>Who</u> <u>was</u> it? *no inversion*
(376) Who <u>was</u> <u>it</u> bothering? *inversion*

Thirdly, our formulation requires fewer *differential* diagnostics to disambiguate between several transformations. These rules are particularly costly because the number of necessary rules grows very quickly with the number of transformations. We have factored the agreement constraints from the *differential* diagnostics. Modularity is a welcome step.

It would be desirable to completely eliminate *differential* diagnostics, rules that mention multiple transformations. We will propose an alternative formulation that achieves many of the same results without the undesirable cost associated with mentioning multiple transformations in a single rule. Traditionally, transformational grammarians imposed ordering constraints to block one rule when another can apply. Marcus' scheme is less restrictive than the traditional ordering constraint: he imposes a *partial* order instead of the more standard *total* order.[183]

Unfortunately, ordering relations are very difficult to formulate, as standard transformational grammarians have discovered. There always seems to be an ordering paradox. An alternative formulation expresses the ordering relation in terms of features.[184] Suppose that imperative requires more precisely determined features than aux-inversion; it cannot trigger while the *tns* features (for example) are underdetermined. Aux-inversion is less restrictive; it will trigger as long as the *tns* features are compatible, whether or not the other possibilities have been excluded. This will assure that aux-inversion takes precedence, without explicitly mentioning both rules in the same diagnostic.

The ordering mechanism is illustrated in (377)-(378). *=?tns* tests for a *pres* or *past* feature, disregarding the other *tns* features; *=tnsless* tests for an uniquely determined *tnsless* feature. A word like *have*, which is both *pres* and *tnsless* ({pres, tnsless}), passes the aux-inversion pattern (377), but fails the imperative pattern, and consequently, aux-inversion will be given first crack. If it should be explicitly blocked (by an agreement constraint), then imperative will be given a chance.[185]

```
(377) (defrule aux-inversion trans
          (pattern (=root) (=?tns =np-) ...)
          (action ...))

(378) (defrule imperative trans
          (pattern (=root) (=tnsless =np-) ...)
          (action ...))
```

In this way, YAP achieves the effects of differential diagnoses without the associated disadvantages. There is a natural separation of performance and competence. The competence idealizations specify agreement constraints; the realistic performance model qualifies them with "ordering" relations. We have proposed a statement of the "ordering" relations which may be more robust than conventional formulations. Nevertheless, the rule ordering problem would completely evaporate if YAP had lexical (side-effect free)

183. A total order is transitive, reflexive, and antisymmetric; every element is ordered with respect to every other. Marcus used a partial ordering scheme (priorities). A partial ordering scheme is not antisymmetric; two elements may have the same priority (unordered).
184. This idea is only partially implemented in the current version, which still contains some differential diagnostics.
185. The *tns* feature is disambiguated when inversion is blocked.

formulations of these transformations. Side-effects should be avoided whenever possible, especially in a deterministic framework.

This chapter has outlined an approach for capturing local structural transformations, taken from Marcus' Parsifal. YAP undoes the transformations by manipulating the lookahead buffer. We have discussed two structural transformations and their interactions. Since it is possible to implement all of Marcus' transformations in this framework, a simple device is adequate for capturing many linguistic generalizations.

8. Wh-movement

A number of long distance transformations are categorized under wh-movement including: wh-questions, embedded questions, relative clauses and topicalization.[186]

(379) Who$_i$ did you see x$_i$? *wh-question*
(380) I wonder who$_i$ you saw x$_i$? *embedded question*
(381) I saw a boy who$_i$ you know x$_i$. *relative clause*
(382) The ball$_i$, Bill took x$_i$. *topicalization*

These constructions are particularly interesting because the trace (x$_i$) can be arbitrarily far from the operator (who$_i$).

(383) Who$_i$ did Bob say that Bill said that ... Mike said I saw x$_i$?
(384) I wonder who$_i$ Bob said that Bill said that ... Mike said I saw x$_i$?
(385) I saw a boy who$_i$ Bob said that Bill said that ... Mike said I saw x$_i$?
(386) The ball$_i$, Bob said that Bill said that ... Mike said I saw$_i$?

Wh-movement illustrates yet another dependency across seemingly unbounded distances. Like there-insertion, the solution is to find a representation (fstructure) where the dependencies are local. YAP has another grammatical role (fwh) to hold the wh-element.[187]

(387) There$_i$ seems x$_i$ likely x$_i$ to seem x$_i$ likely ... *move-np*
(388) Who$_i$ did Bob say that x$_i$ Bill said that x$_i$ ... *move-wh*

There are understood *fwh* elements in (388) just as there are understood *fsubj* elements in (387). The binding relation forms equivalence classes in both cases. The equivalence property is very convenient for computational reasons discussed in chapter 5. All the co-indexed elements are represented collectively as a single node, not once for each individual member. Consequently, wh-movement is bounded in fstructure, even though it appears to have unbounded consequences (see figure 13).

186. Many people object to the topicalization construction.
187. Our *fwh* role is like Bresnan-Kaplan's super-down register, Chomsky's comp node, Marcus' wh-comp feature, Woods' hold cell. Although these mechanisms are similar to one another, they do have slightly different properties. For example, YAP's *fwh* role is passed from *phrase* to phrase whereas the other mechanisms pass the element from *clause* to clause. In this respect, YAP's approach is more like [Koster78] and [Gazdar79a,b,c] which treat all nodes equally; there are no special bounding properties associated with clause nodes.

Fig. 13. Wh-movement
Who$_1$ did Bob say that x_1 Bill said ...

x_1	pred: who
x_2	pred: do
	fwh: x_1
	tns: {past}
	fsubj: x_3
	fxcomp: x_4
x_3	pred: Bob
x_4	pred: say
	fwh: x_1
	tns: {tnsless}
	fsubj: x_3
	fscomp: x_5
x_5	pred: say
	fwh: x_1
	tns: {past}
	fsubj: x_6
x_6	pred: Bill

There are some differences between move-np and move-wh; move-np uses lexical (predicate) rules to bind the intermediate subjects whereas move-wh uses structural (ps) rules to bind the intermediate fwh slots. Compare (389) and (390);[188] Move-wh is a structural rule because it is constrained by phrase structure rules such as (390), whereas move-np is lexical because it is constrained by predicate rules as in (389).

188. It is possible to represent these rules much more efficiently using a markedness theory. For example, the head is unified with its mother (by x-bar theory) unless explicitly marked otherwise.

```
(389) seem-1 -> cxcomp:{ap-, vp-}
      cxcomp = fxcomp(up)
      fsubj(up) = fsubj(fxcomp(up))                move-np

(390) vp -> chead:v (cobj:np-) (cxcomp:xp-)
      chead = up
      cobj = fobj(up)
      cxcomp = fxcomp(up)
      fwh(up) = fwh(fxcomp(up))                    move-wh
```

8.1 Island Phenomena

Wh-elements cannot be extracted from just any phrase; there are certain "islands" which are opaque to wh-movement. Islands are be explained in terms of *consistency* and *coherence* in the Bresnan-Kaplan framework. Some extractions are blocked because the *fwh* slot is already filled (inconsistent) and some are blocked because there isn't a slot to fill (incoherent).

8.1.1 Wh-islands

In general, there can only be one extraction from a phrase because the *fwh* slot only has room for one value; multiple values will be inconsistent. Hence the following sentences are ungrammatical because there are inconsistent fwh elements associated with the bracketed expressions.[189]

(391) *Who$_i$ does John wonder [where Bill saw t$_i$]?
(392) *What$_i$ did you ask me [where you could buy t$_i$]?
(393) *What$_i$ did [who see t$_i$]?
(394) *I wonder what$_i$ [who bought t$_i$]?
(395) *What$_i$ does John wonder [where to put t$_i$]?
(396) *Where$_i$ does John wonder [what to put t$_i$]?
(397) *What$_i$ does John wonder [to put t$_i$ where]?
(398) *Where$_i$ does John wonder [to put what t$_i$]?

189. These examples were given in Ken Hale's 1979 fall class at MIT.

There are some wh-islands which allow extraction. We have no explanation for this fact; YAP cannot currently parse wh-island violations. This is a very marked phenomenon which might be covered by a marked rule.[190]

(399) ?What does John know how to do?
(400) ?What did John ask how to cook?
(401) ?Here are the books that I don't know what to do with?
(402) ?I just read a book which I can't figure out why anyone would write.
(403) ?I like the girl that you wonder what John sees in.
(404) ?I found the book that John couldn't remember what the title of was.

8.1.2 Ross' Complex NP Constraint

[Ross67] observed that extraction is generally blocked by *np-* brackets as in (405)-(407). (This is an over simplification.)

(405) *Who$_i$ do you know [$_{np-}$ the man that married t$_i$]?
(406) *Who$_i$ did you hear [$_{np-}$ a rumor that john betrayed t$_i$]?
(407) *Who$_i$ did you find [$_{np-}$ a copy of a photograph of t$_i$]?

YAP expresses these facts in the *np-* ps-rule. Most ps-rules pass the *fwh* element though constraint equations. For example, the *vp* ps-rule has a constraint equation to pass the *fwh* element into its *xcomp*: *fwh(up) = fwh(xcomp(up))*. There is no such rule associated with *np-*. Hence, an attempt to move an fwh element over an *np-* bracket will be incoherent. This accounts for the minimal contrast between (408)-(410) and the examples above.

(408) Who do you know that John married?
(409) Who did you hear that John betrayed?
(410) Who did you find?

190. These sentences were given in a recent talk by George Hart at MIT. Some informants find these sentences perfectly acceptable while others (including the author) find them extremely marginal.

There are some more difficult cases. For example, if extraction is blocked by *np-*, then why is (411) grammatical? YAP has a marked rule to cover this case. These *picture noun phrases* are still problematic for linguistic analysis. The answer appears to involve the specificity of the *np-*.

(411) Who did you see [$_{np-}$ a picture of t]?
(412) *Who did you see [$_{np-}$ John's picture of t]?

An account has been provided for both types of islands. We do not claim that these facts follow from YAP's design. Our position is much weaker; we merely claim that these facts are compatible with the design. Many linguists are currently working on a more explanatory theory.

8.2 Gap Finding

The really hard problem with wh-movement is finding the "gap" where the wh-element originated. This is not particularly difficult for a non-deterministic competence theory, but it is (probably) impossible for a deterministic processing model. YAP has made some simplifying approximations to the competence idealization which may be valid in a realistic performance model. In an ideal non-deterministic framework, there could be a phrase structure rule like:

```
(413) up:gap-np- -> d1:t
      fwh(up) = d1
```

Unfortunately, it is very difficult to formulate this rule in a deterministic framework. YAP approximates the ideal competence by looking for a gap after the other default ps actions have failed. Find-gap is a new default-ps action which is applied after the other actions as in (414).

```
(414) attach
      predict
      close
      find-gap
```

This heuristic favors the latest possible gap. It corresponds to Fodor's Last-Resort Model of Gap Finding [Fodor78]. As she correctly observes, there are some problems with this model. Like other marked exceptions (see chapter 3), there are some marked rules to handle the problematic cases. Before suggesting some modifications to save the last-resort model, it would be useful to consider some alternatives. Fodor proposed three models of gap finding (415)-(417) and ultimately settled on the third alternative.

(415) First-Resort ([Marcus79])
(416) Last-Resort (YAP)
(417) Lexical Expectation/Arc-Ordering ([Kaplan72], [Fodor78])

The first-resort and last-resort models can be implemented by the default ps actions. The first-resort model orders *find-gap* first whereas the last-resort model orders it last.

(418)

First-Resort	Last-Resort
find-gap	attach
attach	predict
predict	close
close	find-gap

The first-resort and last-resort models do not exclude lexically marked cases; they merely suggest an unmarked default. In some sense, the arc-ordering strategy denies structural correlations; it explicitly lists the preferences for each verb and hence it would be optimal just in case the various structural possibilities were randomly[191] distributed throughout the lexicon.[192] We believe there is a strong bias in favor of (416), although it may be overruled by lexical marking in certain cases. Let us consider some evidence:[193]

8.3 Evidence for the Last-Resort Model

(419) I gave the boy who you wanted to give the books to three books.

Sentence (419) is unacceptable.[194] Grammatically speaking, it is extremely ambiguous; there are no less than four possible gaps as shown in (420).

191. A set is *random* when the shortest description explicitly lists each of its members.
192. Arc-ordering is often formulated within a depth first (DFS) control structure. The DFS is in fact imposing a structural constraint; it encourages low attachment. In Marcus' non-deterministic framework, these structural correlations have to be stated elsewhere. The default ps-actions seem to be a reasonable place.
193. Possible gaps are shown in parentheses. Plus (+) and minus (−) indicate relative processing difficulty. The more acceptable of the pair are marked with a plus.
194. Of the 40 test sentences in [Marcus79, Appendix D], this is the only one that YAP cannot parse. (Some informants find the last gap acceptable as in: *I gave the boy [who you wanted to give the books to t] three books.* This strategy is not incompatible with the Last-Resort Model, although it would require a slight modification.)

(420) # I gave the boy who you wanted (t) to give (t) the books (t) to (t) three books.

Why is it so difficult to find to find the gaps? The last-resort model prefers to attach lexical material over gap finding and hence it misses all the gaps. This unacceptable sentence is very supportive of the last-resort model but rather damaging to the first-resort model which can easily (?!) find the first gap. The examples don't need to be so extreme. We have already seen a garden path sentence (421) also favoring the last-resort model. (422) shows that these GPs are fairly productive.

(421) # I told the boy the dog bit Sue would help him.
(422) ??I called the guy who the car was smashed up by a rotten driver.

Corollary (423)[195] immediately follows from the last-resort model: np gaps are extremely marked[196] in positions immediately before lexical noun phrases. The reason should be obvious; the last-resort model prefers attaching the lexical noun phrase over creating the gap, unless there is positive evidence (i.e. semantic clues) to overrule the default. This corollary accounts for the badness of (421) and (422). Two of the possible gaps in (420) are also excluded under this corollary to the last-resort strategy.

(423) <u>The Trace-NP Corollary</u>: In the unmarked case, #[... t_i NP ...], where t_i is bound to a noun phrase.

This corollary correctly predicts preferences in double object constructions. The lexical noun phrase is generally interpreted as the first object unless there is positive evidence to the contrary. Even then, the marked interpretation is generally less acceptable.[197]

(424) + What did I give the boy t?
+ Who did I give the book to t?
− Who did I give t the book?

(425) + What did you call a drunken sailor t?
− Who did I call t a rotten driver?

195. The corollary has been stated as a processing filter quite analogous to the competence filters of [Chomsky and Lasnik77]. Filters are a convenient method of *describing* the facts, but they are probably inadequate as *explanations*. In this case, we cannot explain why last-resort seems to be the unmarked case.
196. There are at least three productive "counter-examples" to the corollary where the filter is inoperative. We will turn to these cases soon.
197. The marked interpretation is excluded from certain dialects.

(426) + What do I consider John t?
 – Who do I consider t a fool?

(427) + What did I tell the boy t?
 + Who did I tell the story to t?
 – Who did I tell t the story?

The last-resort strategy is consistent with the Trace-X Filter (428), which is similar to constraint (429).[198] The constraint predicts that a trace of category X cannot appear just before lexical material of category X. Sentences (424)-(427) are consistent with this generalization of the Trace-NP Corollary. Unfortunately, there is little evidence in English to justify the move away from the Trace-NP Corollary. (The crucial evidence comes from French.)

(428) The trace-X Filter: In the unmarked case, #[... t_i X ...], where t_i is a trace of category X.

(429) The XX Extraction Constraint: If at some point in its derivation a sentence contains a sequence of two constituents of the same formal type, either of which could be moved or deleted by a transformation, the transformation may not apply to the first constituent in the sequence. [Hankamer73].

Although the last-resort strategy has many of the right characteristics, there are also many problems which require marked rules. We will consider the following three problems here:[199]

(430) Ambiguity
(431) Lexical Marking
(432) Length

198. Hankamer proposed that the XX Extraction Constraint belongs in competence. Since it can be violated (in the marked case), we prefer to place it in performance. [Fodor78] also views the constraint as a processing matter.
199. It has been suggested that cleft sentences like, *What I wanted that for t nobody could understand*, form another class of marked exceptions to the performance filter.

8.3.1 Ambiguity

There are some ambiguous sentences which strongly resemble the pseudo-attachment case. In the pseudo-attachment case, there is a lexical *xp-* with two possible mothers. Pseudo-gap is exactly analogous except the *xp-* is a trace.

(433) Put the block in the box on the table. *pseudo-attachment*

(434) Who do you want (t) to eat (t)? *pseudo-gap*
(435) The duck is too old (t) to eat (t).
(436) Who did Mary promise (t) that she would marry (t)?
(437) To whom did Father say (t) that he was planning to write (t)?
(438) Where did he say (t) he was going (t)?
(439) When did he say (t) he was going (t)?

Only (434) has been implemented, though the others shouldn't be much more difficult. Pseudo-gaps have many of the same problems as pseudo-attachment. It is (probably) impossible to find all the gaps in sentences like (440). YAP settles for the first and last possible gaps as in (441), in the absence of disambiguating information.

(440) Who do you want (t) to want (t) ... (t) to want (t) to eat (t)?
(441) Who do you want (t) to want ... to want to eat (t)?

8.3.2 Lexical Marking

The unmarked case can be overruled by the lexicon as in (443). These cases have not been implemented.

(442) + Who did the teacher walk to the cafeteria with? *unmarked*
– Who did the teacher walk to the cafeteria?

(443) – Which book did the teacher read to the children from? *lexically marked*
+ Which book did the teacher read to the children?

Even though *read* and *walk* have the same subcategorization features (they both select an optional object and a verbal complement), they have different preferences as illustrated by (442) and (443). This evidence is often taken to support the arc-ordering position. Although we accept lexically marked preferences, there are other implications associated with that position which are incompatible with the framework presented here; in particular, arc-ordering is crucially non-deterministic.[200]

8.3.3 Length

Notice that judgments are less and less sharp as the second object increases in length. This is completely unexplained by our account. There are other length phenomena (such as heavy np shift) which are more widely accepted. We seem to be missing a generalization. However it isn't clear how to capture the length phenomena. [Frazier and Fodor78] used a front end filter (PPP) which divided chucks into roughly six words. Although this is an interesting proposal, it isn't clear how it could be implemented.

200. [Rich75] gives a critical review of the arc-ordering position. In his opinion:

Linguistic Phenomenon	Computational Mechanism	Assessment
Center-embedding	single-place HOLD list	wrong
Preferred readings of Ambiguous Sentences	ordered trying of alternatives (arcs)	inadequate
GP sentences	back-tracking	somewhat right
Perceived Complexity Differences	HOLD list costing arc counting	inconclusive

His arguments are very convincing. One could view YAP as a DFS which only backs up after it takes a very serious GP. (We haven't implemented a GP recovery procedure yet, but backup would be the easiest way to do so.) A sentence is unacceptable just in case it causes YAP (as modified) to backup. This is a precise definition. The problem with the arc-ordering position is that backup describes both crashingly unacceptable GPs and extremely subtle preferences of ambiguous sentences. The sharpness is not related to any measure of backup that has been proposed. We suggest that subtle preferences have a very different explanation from GPs.

(444) # Who did you call t it?
???Who did you call t that?
??Who did you call t a rotten driver?
?Who did you call t the worst driver that you ever ...

8.4 Summary

We have discussed four cases of wh-movement: wh-questions, embedded questions, relative clauses, and topicalization. Movement constructions suggest some interesting topics in both competence and performance.

(445) Competence: locality principles & island phenomena
(446) Performance: gap finding

We have shown that "unbounded" movement phenomena are local using an appropriate representation, such as Bresnan-Kaplan's fstructure.[201] Locality is extremely convenient for processing because it enables YAP to apply movement rules *without approximation.* If the rules were truly non-local they would require unbounded memory and hence we should expect to discover empirical discrepancies from the competence idealization. However, since the idealization is local, there need not be any empirical discrepancies.

The locality issues are extremely complex; we have only addressed a few cases. Much of the linguistic discussion deals with islands which are opaque to wh-movement. These islands should have a natural formulation in our representation (fstructure or move-alpha*). We have given an account (more or less) for two types of islands: wh-islands and Ross' Complex NP Constraint. This is still an active area of linguistic inquiry.

201. It also is possible to represent movement locally in Chomsky's framework, using equivalence classes. We have previously suggested that Bresnan-Kaplan's merge operator (=) is an equivalence relation. All the nodes which have been merged together (co-indexed) form a single equivalence class (index), which is represented as a single node in fstructure. For example, in the raising case (move-np), all the understood subjects are co-indexed into a single node in fstructure. Similarly co-indexed traces in comp (*/wh* in YAP) are also a single node in fstructure.

Using the same basic approach, we could represent movement locally in Chomsky's system. Let move-alpha be a relation between two phrases, and let move-alpha* be the transitive, symmetric and reflexive closure of move-alpha. Move-alpha* is similar to Bresnan-Kaplan's merge (=) operator; it too defines equivalence classes corresponding to the index. The claim that movement is local in fstructure corresponds to a claim that movement is local on indexes (equivalence classes under move-alpha*).

The most difficult problem is finding the gap. We have argued for a last-resort model. It is consistent with some garden path data and Hankamer's XX Extraction Constraint, although it does have some problems. The most serious problem is lexical marking. It was suggested that marked rules could apply in the crucial cases, although the proposal has not been implemented. There also appear to be some length effects, which are also unexplained. We outlined a partial solution to the pseudo-gap phenomena.

Despite these problems, we have implemented a simple device which captures many of the wh-movement phenomena. This result[202] considerably weakens the traditional view that processors must be Turing Equivalent. The next chapter will illustrate a "simple" mechanism for parsing many conjunction phenomena, which were also believed to require inordinate resources.

202. Many other researchers have designed "simple" devices to capture wh-movement. See [Marcus79] and [Gazdar79a,b,c] for two examples.

9. Conjunction

Conjunction has been one of the most difficult constructions to parse because there seem to be so many possible alternatives. Conjunction is a very good test of the FS hypothesis. How can we approximate the ideal competence model so that a FS processor can parse conjunction? We've made some impressive initial progress, although there is still substantial work to be done.

9.1 Simplifying Assumptions

Many parsers have found conjunction difficult because they consider too many possibilities. It is extremely important to consider as few alternatives as possible. We will impose several very strict limitations on conjunction in order to limit the scope of the problem. All of these restrictions are controversial.

9.1.1 The Constituent Assumption

(447) Assumption: Conjunction applies to constituents, not to arbitrary fragments.

(448) The scene [of the movie] and [of the play] was in Chicago.
Which [boys] and [girls] went?
[Which boys] and [which girls] went?
Which boys [went to the ball] and [took the jar]?

Although (447) is generally accepted, there have been some objections. Sentences like (449)-(450) have been used to argue that conjuncts may not always be constituents. We will argue that despite appearances both (451) and (452) are constituents.

(449) John [drove through] and [completely demolished] a plate glass window. [Woods73]
(450) Mary [expressed costs in dollars] and [weights in pounds]. [Martin80]

(451) $[_{vp}$ drove through $[_{np-}$]]
(452) $[_{vp}$ $[_{v}$] weights in pounds]

The constituent assumption is very convenient for processing, as we will see.

9.1.2 The Category Assumption

(453) Assumption: Each conjunct has the same category.

This assumption is also fairly standard, though there have been arguments to the contrary. [Martin80] provides the following "counter-example". (455) is his analysis; (456) is our own.

(454) We expect difficulties now and in the future.
(455) We expect difficulties $[_{np\text{-}}$ now] and $[_{pp\text{-}}$ in the future]. *Martin*
(456) We expect difficulties $[_{pp\text{-}}$ now] and $[_{pp\text{-}}$ in the future]. *YAP*

In this case, it seems reasonable to call *now* a prepositional phrase. This is a small cost to pay to save the category assumption.

9.1.3 The Across-the-Board Convention

(457) Assumption: Each conjunct has the same number of wh-gaps. Furthermore, the gaps have the same category.[203]

The last three assumptions can be summarized in Gazdar-Notation[204] as (458).[205] (The comparative construction illustrates the need for some more categories (q, qp and qp-) to represent quantifiers. Comparatives have not been implemented.)

(458) Assumption: Each conjunct has the same Gazdar-Notation.

(459) *John is easy $[_{vp\text{-}/np\text{-}}$ to please] and $[_{vp\text{-}}$ to love Mary].
John is easy $[_{vp\text{-}/np\text{-}}$ to please] and $[_{vp\text{-}/np\text{-}}$ to love].

(460) *The man who $[_{s/np\text{-}}$ Mary loves] and $[_{s}$ Sally hates George] computed my tax.
The man who $[_{s/np\text{-}}$ Mary loves] and $[_{s/np\text{-}}$ Sally hates] computed my tax.

(461) The kennel which $[_{s/np\text{-}}$ Mary made] and $[_{s/np\text{-}}$ Fido sleeps in] has been stolen.
The kennel in which $[_{s/pp\text{-}}$ Mary keeps drugs and $[_{s/pp\text{-}}$ Fido sleeps] has been stolen.

203. It seems that the gaps have to be identical in every respect, not just category. That is, they have the same reference, person, number, gender, case, inflection, etc.
204. In Gazdar-Notation, *X/Y* refers to a node of category *X* containing a gap of category *Y*.
205. These examples are taken from [Gazdar79c]. Tough movement and comparative are not currently implemented.

*The kennel (in) which [$_{s/np-}$ Mary made] and [$_{s/pp-}$ Fido sleeps] has been stolen.

(462) John saw more horses than [$_{s/np-}$ Bill saw] or [$_{s/np-}$ Pete talked to].
John saw more horses than [$_{s/qp-}$ Bill saw cows] or [$_{s/qp-}$ Pete talked to cats].
*John saw more horses than [$_{s/qp}$ Bill saw cows] or [$_{s/np-}$ Pete talked to].

9.2 Simple Cases

In the simple case, the conjuncts happen to be in up1 and down2 as below.

(463) Bob and Bill saw it.

(464) [$_s$ Bob]
[$_{np-}$ Bob] *first conjunct*
==WALL==
[$_{conj}$ and]
[$_{np-}$ Bill] *second conjunct*
[$_v$ saw]

Conjunction is possible in (464) because down1 is a conjunction and up1 and down2 are constituents of the same category with matching gaps. There is a marked rule which looks for this pattern.

9.2.1 Attaching Conjuncts

Attaching conjuncts is different from other types of attachment; there is a special slot in cstructure nodes for conjuncts.

(465) np- -> np- conj np- *standard*
(466) [$_s$ [$_{np-}$ [$_{np-}$ Bob] and [$_{np-}$ Bill]] ...

(467) np- -> chead:np cxcomp:{vp-} cxcomp:{s-} cconjuncts:np- *YAP*
(468) [$_s$ [$_{np-}$ Bob and [$_{np-}$ Bill]] ...

Using the standard approach, YAP couldn't attach *Bob* to the root because there might be a conjunction node in between. Consequently, attachment wouldn't be possible until the right edge has been read. But this would prevent early closure (A-over-A closure principle) because no node could be attached until all of its descendants have been completed. This is very unfortunate. YAP's approach avoids this problem because there are no nodes between the first conjunct *Bob* and the root, and hence, attachment is possible before conjunction is considered.[206]

After attaching the conjuncts, [$_{np-}$ Bill] will fill the *cconjuncts* slot of [$_{np-}$ Bob] and the machine state will be:

(469) [$_S$ Bob and Bill]
[$_{np-}$ Bill]
==WALL==
[$_V$ saw]
[$_{np-}$ it]
[$_{punct}$.]

The sentence will now be parsed as if [$_{np}$ Bill] is the subject.[207]

9.2.2 Attention Shift

The approach just outlined works on (470), but fails on (471) where minimal attachment is misleading.

Fig. 14. Attention Shift
sentence: I saw Bob and Bill saw me.
input pointer: me.

before	after
[$_S$ I saw Bob]	[$_S$ I saw Bob]
[$_{vp}$ saw Bob]	[$_{vp}$ saw Bob]
[$_{np-}$ Bob]	[$_{np-}$ Bob]
==WALL==	[$_{conj}$ and]
[$_{conj}$ and]	==WALL==
[$_{np-}$ Bill]	[$_{np-}$ Bill]
[$_V$ saw]	[$_V$ saw]

206. Yet another alternative would use the standard phrase structure rules. It would attach the first conjunct as if there were going to be a conjunction. The second conjunct would then be Chomsky Adjoined when it is discovered. (This may be a notational variant of the current implementation.)
207. There is only one difference; [$_{np-}$ Bob and Bill] is plural whereas [$_{np-}$ Bill] is singular. YAP's solution assumes that all of functional features are inherited. In fact, number values are not inherited in the usual way. YAP actually replaces the number value in this case. There is a more attractive solution to be found.

(470) I saw Bob and Bill.
(471) I saw Bob and Bill saw me.

The solution is to shift the attention[208] of the machine past the *and* building *Bill saw me* bottom up. Then the machine will return its attention back to the conjunction and finish the sentence as if [$_S$ Bill saw me] came prepackaged as a single unit.

YAP shifts its attention by moving down1 into the upper buffer as in figure 14. Attention return is just the inverse; YAP moves up1 back into the lower buffer.[209] The technique is very general; it allows bottom-up chunks to appear prepackaged. Attention shifting is heavily used to parse noun phrases.

There are some important issues concerning the order of attention shift and return in the default ps rules. Return is last. It isn't clear where shift should be; Marcus ordered it first,[210] we've ordered it much later. The issues are not well understood; we're not prepared to make a coherent argument.[211]

(472)

YAP	Marcus' Parsifal
attach	attention shift
predict	find gap
attention shift	attach
close	predict
find gap	close
attention return	attention return

208. The terminology is taken from [Marcus79] who used a similar technique to parse noun phrases.
209. There are two registers associated with each node (*as-status* and *as-return-status*) which prevent infinite attention shifts and returns. The details aren't very interesting.
210. Marcus' attention shift mechanism was conditional on category type. Parsifal would attention shift for noun phrases, but not for verb phrases or prepositional phrases. Our mechanism applies to all categories.
211. The ordering of actions in Marcus' Parsifal is partly defined by the interpreter (attention shift and return) and partly implicit in the grammar (attach, predict, close and find gap). The implicit order may be incorrect; it is our own interpretation of his grammar.

9.2.3 Closing

After attention shifting to parse $[_{s}$ Bill saw me], the machine state is (473) (left side). The machine will then close up1 repeatedly until conjunction is possible.

(473)

before	after closing once	after closing twice
$[_{s}$ I saw Bob]		
$[_{vp}$ saw Bob]	$[_{s}$ I saw Bob]	
$[_{np-}$ Bob]	$[_{vp}$ saw Bob]	$[_{s}$ I saw Bob]
==WALL==	==WALL==	==WALL==
$[_{conj}$ and]	$[_{conj}$ and]	$[_{conj}$ and]
$[_{s}$]	$[_{s}$]	$[_{s}$]
$[_{np-}$ Bill]	$[_{np-}$ Bill]	$[_{np-}$ Bill]
$[_{v}$ saw]	$[_{v}$ saw]	$[_{v}$ saw]

Conjunction applies just as it did in the simple case, *I saw Bob and Bill.* Down2 fills the *cconjuncts* slot of up1, leaving the machine in (474). The rest of the sentence parses just like the simple sentence, *Bill saw me.*

(474) $[_{s}$]
==WALL==
$[_{np-}$ Bill]
$[_{v}$ saw]

9.2.4 Summary of the Simple Cases

Let us summarize the simple conjunction rule. First, the machine attention shifts for the non-minimal attachment case (476). In the non-minimal case, YAP will predict an *s* just before *Bill saw me.* Then YAP will return attention to the *and.*

(475) I saw Bob and Bill.
(476) I saw Bob and Bill saw me.

Secondly, YAP tries to attach conjuncts, if possible. Up1 and down2 have to be constituents and should match in category and gaps. Finally, if that doesn't work, YAP will close up1.

(477) Attention shift
(478) Attach-conjuncts
(479) Close

This approach has some problems. It finds only the lowest attachment, not the full range of ambiguous possibilities. YAP should pseudo-attach conjuncts in ambiguous cases such as (480). It should be possible to implement pseudo-attachment in these cases, but the details have not been worked out.

(480) Bill told Bob [that Mike told Harry] and [Sam told Jack].

There are more difficult cases where pseudo-attachment is not a likely solution. It is not clear how (482) and (483) could be represented in a single structure. Even worse, YAP prefers the unlikely interpretation (483) because *Bill left* builds a clause bottom-up.

(481) I know Bob and Bill left.
(482) I know [Bob and Bill] left
(483) [I know Bob] and [Bill left]

The general approach has been very effective although there are many problems to be solved.

9.3 Deletions

It is possible for one of the conjuncts to contain a deleted element. In the *gapping* case, the verb in the second conjunct is deleted; in *right node raising*, an object in the first conjunct is missing.

(484) Bob saw Bill and Sue Mary. *gapping*
(485) Bob looked at and Bill took the jar. *right node raising*

Both of these constructions appear to violate the constituent assumption. With a deletion analysis, though, it is possible to save the constituent assumption. As we have suggested, (484)-(485) will be analyzed as:[212]

212. Right node raising is usually analyzed as:

[Bob looked at t_i] and [Bill took t_i] [the jar]$_i$

Our analysis is simpler to implement. Although this alone isn't a valid reason to prefer one analysis over another, there is sufficient controversy over right node raising that it didn't seem worth the effort to implement it precisely.

(486) [Bob saw Bill] and [$_S$ Sue [$_V$] Mary]
(487) [Bob looked at [$_{np-}$]] and [Bill took the jar]

9.3.1 Right Node Raising

YAP has a marked rule to parse right node raising. When there is a conjunction (e.g. *and*) in down1 and up1 can't close, then YAP assumes right node raising. The analysis crucially depends on the constituent assumption; if a conjunct is not a complete constituent, then by assumption the rest must have been deleted. Having detected the deletion, YAP undoes the transformation, inserting an empty noun phrase back into the buffer as in figure 15.

The analysis has some problems; it does not bind the empty noun phrase to an object in the second conjunct. YAP would erroneously accept ill-formed sentences such as (488). There is some controversy over the appropriate binding mechanism; it isn't clear if it is movement as in [Gazdar79c] or anaphoric.[213]

Fig. 15. Right Node Raising
sentence: Bob looked at and Bill took the jar

before	after
[$_S$ Bob looked at]	[$_S$ Bob looked at]
[$_{vp}$ looked at]	[$_{vp}$ looked at]
= =WALL= =	= =WALL= =
[$_{conj}$ and]	[$_{np-}$]
[$_{np-}$ Bill]	[$_{conj}$ and]
[$_V$ took]	[$_{np-}$ Bill]
	[$_V$ took]

213. It is generally agreed that the subject of *drink* is anaphorically bound in the following cases.

Drinking gin can be fun.
It doesn't require a glass to drink gin.
Having drunk gin all day, I was completely wasted.

There are several important differences between anaphoric control and movement. This paper though will not discuss bound anaphora.

(488) *I took and you went.

Optional arguments illustrate another problem. YAP will detect only obligatory elements which have been deleted; optional elements are also subject to deletion. YAP will not detect an object of *ate* in (489).

(489) I ate ($[_{np-}$]) and you drank everything they brought.

9.3.2 Gapping

"Gapping" is the case where the second conjunct's verb has been deleted. (490) is a simple example.

(490) [Bob saw Bill] and $[_s$ Sue $[_v$] Mary]

YAP parses these by undoing the transformation. When the lower buffer contains a conjunction followed by two noun phrases, YAP inserts an empty verb into the buffer. To exclude interpretations such as (492), YAP merges the predicates from both conjuncts. See figure 16.

Fig. 16. Gapping

sentence: Bob saw Bill and Sue Mary.

before transformation

```
[s Bob saw Bill]
==WALL==
[conj and]
[np- Sue]
[np- Mary]
```

after transformation

```
[s Bob saw Bill]
==WALL==
[conj and]
[np- Sue]
[v ]                see-1 -> fsubj:np- fobj:np-
[np- Mary]
```

(491) Bob persuaded Bill to leave and Sue Mary.
(492) *Bob persuaded Bill to leave and (*Bob persuaded*) Sue Mary.

The implementation is not as general as it should be; the verb can be deleted in many other contexts. YAP can find a deleted verb in any projection of v (in vp, vp-, s and s-). For example, YAP correctly parses (493). Unfortunately, it finds only the lowest possible interpretation; it will not discover (494) unless there is some positive reason (i.e. semantics) to reject (493). The gapping pattern crucially depends on two noun phrases; it will not detect gapping when the second object is an xcomp as in (495)-(497). Aside from the ambiguity problem, these problems shouldn't be too difficult to correct. The simple cases of gapping were implemented to show plausibility within our restricted framework.

(493) Bob [gave Bill a ball] and [$_{vp}$ Sam a jar].
(494) [Bob gave Bill a ball] and [$_{s}$ Sam a jar.]

(495) Bob persuaded Bill to leave and Sam [$_{vp-}$ to stay].
(496) I expressed costs in dollars and weights [$_{pp-}$ in pounds].
(497) I considered Bill likely to win and Sam [$_{ap-}$ likely to lose].

9.4 Summary

In summary, we have presented a simple approach to parse many conjunction constructions including some cases of right node raising and gapping. Although there are many problems to be solved, these analyses indicate that it is plausible for a FS deterministic processor to parse conjunction. This discussion responds to the traditional arguments that a FS processor cannot *in principle* capture the conjunction generalizations.

We have previously suggested that closure actually simplifies conjunction. YAP uses closure to find the first conjunct; it will continuously close off up1 until the first conjunct is in up1. Furthermore, closure assures that all *possible* conjuncts will be in the upper buffer; this makes it much easier to pseudo-attach conjuncts since it is easy to find all the possibilities.[214]

214. YAP does not currently pseudo-attach conjuncts, although it was designed with this in mind.

10. Conclusion

We have hypothesized that a computationally simple device is sufficient for processing natural language. By incorporating two processing constraints, FS and Marcus' Determinism, it was possible to construct a parser which approximates many competence idealizations. YAP was designed to fail precisely where the idealizations require unrealistic resources. YAP's success, as far as it goes, provides some evidence for the hypothesis.

10.1 The Traditional Position

Traditionally there have been many arguments for computationally complex models of natural language. Much of the early literature, though, does *not* refute our hypothesis, but merely cast doubt on its feasibility. Admittedly, it is easier to find descriptions using more powerful (complex) techniques, but is it *necessary* to use more powerful techniques? The traditional arguments are extremely negative; if the problem is really as hard as they suggest, then the only solution is to grin and bear it. It is easy to show how hard a problem might be, but it is a real accomplishment to find a simple elegant solution.

Chomsky's early arguments are rightly cautious; they do not exclude the possibility of a FS processor. He criticizes contemporary FS approaches as inelegant, and then proposes a computationally complex alternative as more *revealing*. Over the years, however, his position has been misinterpreted as a complete refutation of FS approaches. It is merely a feasibility argument. To a certain extent he is correct, [Chomsky56, pp. 113] "the grammar of English is materially simplified if phrase structure description is limited to a kernel of simple sentences from which all other sentences are constructed by repeated transformations; and that this view of linguistic structure gives a certain insight into the use and understanding of language." Hence, competence idealizations should use powerful devices. However, this does *not* say that language should be processed by *exactly* the same machinery.

This is a very common situation in engineering. Engineers develop ideal models to gain fruitful insights; they do not expect their model to perfectly replicate the real world. They will use the theory as far as it goes, and then joke about "Murphy's Law". Idealizations are very useful, but they can't be taken too seriously; they simply don't work in all cases. Physical machines do not behave ideally.

[Chomsky56] provides a "counter-example" to FS models. It generates arbitrary center-embedding and hence it is beyond the generative capacity of a FSM. Since his counter-examples are grammatical (part of the ideal competence model of language), this *proves* that a FSM cannot process competence.[215] However, it is well-known that arbitrarily deep center-embedding is universally unacceptable, and hence, Chomsky's arguments do *not* apply to performance. We have no reason to exclude the possibility of a FS parser.

He correctly suggests that a parser should encode a *simple and "revealing" grammar.* It is not clear how this can be accomplished with a simple device. YAP introduces a number of approximations (i.e. bounded stack, finite lookahead, ...) in order to approximate an elegant (though complex) competence grammar with reasonable resources. Chomsky has questioned this move for two reasons:[216]

215. "Turning now to English, we find that there are infinite sets of sentences that have dependency sets with more than any fixed number of terms. For example, let S_1, S_2, ... be declarative sentences. Then the following are all English sentences:

(13) (i) If S_1, then S_2.
(ii) Either S_3, or S_4.
(iii) The man who said that S_5, is arriving today.

These sentences have dependencies between 'if'-'then', 'either'-'or', 'man'-'is'. But we can choose S_1, S_3, S_5 which appear between the interdependent words, as (13i), (13ii), or (13iii) themselves." [Chomsky56 pp. 115]

216. "Although we have found that no finite-state Markov process [YAP] that produces sentences from left to right can serve as an English grammar [competence], we might inquire into the possibility of constructing a sequence of such devices that in some nontrivial way, come closer and closer to matching the output of a satisfactory English grammar. Suppose, for example, that for fixed *n* we construct a finite-state grammar in the following manner: one state of the grammar is associated with each sequence of English words of length *n* [ordered by statistical frequency] ... as *n* increases, the output of such grammars will come to look more and more like English ... This fact has occasionally led to the suggestion that a theory of linguistic structure might be fashioned on such a model ...

Whatever the other interests of statistical approximation in this sense may be, it is clear that it can shed no light on the problems of grammar. There is no general relation between the frequency of a string (or its component parts) and its grammaticalness ... there is no significant correlation between order of approximation and grammaticalness. If we order the strings of a given length in terms of order of approximations to English, we shall find both grammatical and ungrammatical strings scattered throughout the list, from top to bottom. Hence the notion of statistical approximation appears to be irrelevant to grammar." [Chomsky56 pp. 116]

(498) Are the approximations revealing?
(499) What are reasonable approximations?

We have attempted to respond to both points. First, they are revealing because they suggest a number of crucial differences between competence and performance. For example, Lasnik's Noncoreference Rule is an impractical idealization; a more realistic approximation (using the A-over-A early closure principle) predicts certain coreferential possibilities which may actually reflect the real empirical facts more accurately than Lasnik's idealization. We have discussed many other constructions which are similar in this respect, such as: center-embedding, crossed dependencies and garden paths.

Chomsky's second criticism is also well-taken; it is very difficult to find independently motivated approximations. He rightly criticizes a statistical approach for missing the relevant generalizations. In this work, we have attempted to motivate effective approximations without sacrificing linguistic significance. YAP captures many linguistic generalizations such as: raising, passive, there-insertion (chapter 6), inversion, imperative (chapter 7), wh-movement (chapter 8), and conjunction (chapter 9).[217] These generalizations are basically orthogonal to the two processing approximations: FS and determinism. Hence, the approach taken here may be a reasonable compromise between processing complexity and linguistic elegance.

10.2 Summary

We have been most concerned with two performance constraints: FS and determinism. Both of these constraints reduce the computational power, which is always a welcome step in computer science. The question is whether the machine retains enough power to parse language. We have demonstrated, by implementing YAP, that it is sufficient to parse certain difficult constructions. Furthermore we have defended a number of simplifying assumptions as more accurate descriptions of the empirical facts. Chapters 1 through 4 discussed some evidence involving center-embedding, crossing dependencies and noncoreference. These constructions are provably complex (in competence), and as predicted, they do not behave ideally, even at severely shallow depths. This is suggestive evidence in favor of our simplifying assumptions. It appears that *all* examples of complex behavior are universally *unacceptable*.

217. One could rightly criticize these transformations as mere stipulations. A truly revealing theory would explain the facts. We have described (stipulated) many of Bresnan-Kaplan's analyses as they are. When deeper explanations are found, it may be worthwhile to redesign YAP.

There are many difficult issues dealing with a particular implementation of the approximations. Chapter 2 discussed several *closure* proposals. We finally settled on a compromise (the A-over-A early closure principle) which has some of the right limiting properties (w.r.t. premature/ineffective), but may have some problems in certain borderline cases (three deep center-embedded sentences). The limiting cases are far more important; the field may not have progressed sufficiently far to make the subtle distinctions necessary for the borderline cases.

Chapter 4 dealt with *attachment strategies*. We advocated a default mode of operation (attach, predict, and then close) which covers most cases although there are many exceptions. The exceptions fall into four classes: early closure (chapter 2), non-minimal attachment (chapter 4), pseudo-attachment (chapter 4) and transformations (chapters 6-9).

Pseudo-attachment illustrates the *delayed binding* approach which is a recurrent theme in this work. The idea is to avoid making decisions which may have to be taken back at a later time. This is particularly crucial in a deterministic framework which prevents the system from reverting previous commitments. In the pseudo-attachment case, the system can decide that it cannot decide how to attach, and hence it attaches both ways.

The delayed binding approach is also central to feature manipulation (chapter 5). An alternative approach would try each feature value combination non-deterministically until it found a combination which doesn't violate any agreement constraints. This can be very time consuming. YAP's approach is a constraint propagation technique; it applies the constraints themselves to the fstructure. The difference between the two approaches becomes apparent when the constraints underdetermine the final outcome, such as (500)-(501). YAP makes a single deterministic pass; it is no harder to search an underdetermined fstructure than any other. A non-deterministic parser, on the other hand, has to search the fstructure once for each combination of values; the underdetermined case requires much more time because there are more combinations of values.

(500) I <u>put</u> it down. *underdetermined tense*
(501) The <u>deer</u> left. *underdetermined number*

The lexicalist position is very compatible with a delayed binding approach. Although it is possible to write a deterministic transformational grammar (as Marcus did), we have found the lexicalist position more sympathetic with the notion of constraints, which is crucial in our formulation of delayed binding. For example, both approaches have a mechanism for "raising" understood subjects as in (502); Bresnan and

Kaplan use the constraint equation, ***subj(up) = subj(xcomp(up))***, where Chomsky uses the transformation *move-np*. Bresnan-Kaplan's constraint equations fall rather naturally into a constraint propagation framework; it might require some ingenuity to reformulate Chomsky's movement as a constraint. Although it is probably possible to reformulate movement in this way, Bresnan-Kaplan's formulation requires little modification to fit into a constraint propagation framework.

(502) John_i seems x_i to be a nice guy.

In summary, we have proposed that a deterministic FS parser is sufficient to parse natural language without sacrificing linguistic generalizations. To justify this claim, we have designed yet another parser (YAP) which encodes many of Bresnan-Kaplan's analyses in a deterministic FS framework. Although there are many unsolved problems (i.e. lexical ambiguity, syntactic/semantic interaction, ...), we have demonstrated plausibility for the underlying design which incorporates both performance (FS and determinism) and competence (Bresnan-Kaplan's lexical framework).

Appendix I - Some Results

Sentences (503)-(536) were taken from Appendix D [Marcus79]. These examples illustrate passive, raising, there-insertion, some lexical ambiguity (*that*, *meet* and *schedule*), aux-inversion, imperative and wh-movement. YAP can parse all of them except (534) which is unacceptable. Chapter 8 discusses this sentence in more detail.

(503) I told that boy that boys should do it.	*that diagnostic*
(504) The jar seems to be broken.	*passive, subject raising*
(505) There seems to be a jar broken.	*there-insertion*
(506) I wanted John to do it.	
(507) I want to do it .	
(508) I persuaded John to do it.	*object rasing*
(509) There seems to have been a meeting scheduled for Friday.	
(510) Schedule a meeting for Friday.	*imperative*
(511) Is there a meeting scheduled for Friday?	*aux-inversion*
(512) A meeting seems to have been scheduled for Friday.	
(513) I told the boy that i saw Sue.	
(514) I told Sue you would schedule the meeting.	
(515) I told the girl that you would schedule the meeting.	
(516) The boy who wanted to meet you scheduled the meeting.	*wh-movement*
(517) The boy who you met scheduled the meeting.	
(518) Who did John see?	
(519) Who broke the jar?	
(520) What did Bob give to Sue?	
(521) Who did Bob give the book?	
(522) Who did Bob give the book to?	
(523) I promised John to do it.	
(524) Who did you say that Bill told?	
(525) You promised to give the book to John.	
(526) Who did you promise to give the book to?	
(527) Who did you promise to schedule the meeting?	
(528) Who did you say scheduled the meeting?	
(529) Who did you persuade to do it?	
(530) What did you give Sue yesterday?	
(531) Who did you ask to schedule the meeting?	

(532) Who do you want to give a book to tomorrow?
(533) Who did you want to give a book to Sue?
(534) # I gave the boy who you wanted to give the books to the books?
(535) Who did you promise to give the book to tomorrow?
(536) Who did you promise to give the book to Sue tomorrow?

YAP can also parse the following conjunction sentences. These sentences were selected to illustrate YAP's abilities, both positive and negative. Many of these sentences may be unacceptable and/or ungrammatical for reasons which YAP does not consider. For example, YAP does no pragmatic analysis; (540) is syntactically well-formed even though it may sound somewhat odd. Similarly, (541) is probably ungrammatical because the trace has conflicting case; it receives objective case from the first conjunct and oblique case from the second. It would be simple enough to change the grammar accordingly. Finally, (542) demonstrates a real problem with YAP's formulation of right node raising; YAP does not require the missing noun phrase to "match" with the right most noun phrase in the second conjunct. Although there are some problems with YAP's formulation of conjunction, it demonstrates some real progress.

(537) Which boys and girls went?
(538) Which boys and which girls went?
(539) Which boys went to the ball and took the jar?
(540) Which boys went to the ball and into it?
(541) What boy did bill look at and give a ball to?
(542) Bob looked at and gave a ball to the boy.
(543) Bob gave Bill a ball and John a jar.
(544) Bob saw Bill and Sue Mary.
(545) I want Bill, Bob, and John to be nice.

The following sentences were taken from a homework problem given by Ken Hale last fall. The first set are all grammatical; the second violate island conditions and, hence are ungrammatical. YAP can parse all the grammatical ones and none of the ungrammatical ones. See the discussion of island phenomena in chapter 8.

(546) Who should I ask where I can get a copy of Aspects?
(547) What is it expected that Max will work on next?
(548) What do you expect that Max will work on next?
(549) What is Max expected to work on next?
(550) What do you expect Max to work on next?
(551) Who is expected to work on case-marking next?

(552) Who saw what?
(553) I wonder who bought what.
(554) John wonders where to put what.
(555) John wonders what to put where.
(556) What does John want to put where?
(557) Where does John want to put what?
(558) Who did you find a photograph of?
(559) It is believed John has won the election.
(560) John is believed to have won the election.

(561) *Who does John wonder where Bill saw?
(562) *What did you ask me where you could buy?
(563) *What is expected that Max will work on next?
(564) *What is expected Max to work on next?
(565) *What did who see?
(566) *I wonder what who bought?
(567) *What does John wonder where to put?
(568) *Where does John wonder what to put?
(569) *What does John wonder to put where?
(570) *Where does John wonder to put what?
(571) *Who do you know the man that married?
(572) *Who did you hear a rumor that John betrayed?
(573) *Who did you find a copy of a photograph of?
(574) *John is believed has won the election.
(575) *John seems won the election.

The following illustrate some other generalizations:

(576) It seems likely that John would be sitting. *it-extraposition*
(577) There seems to be a table in the kitchen. *there-insertion*

(578) That I might take a ball seems likely. *sentential subjects*
(579) For me to take a ball seems nice.
(580) To take a ball seems nice.

(581) I wonder what to do. *embedded questions*
(582) I wonder what he should do.
(583) I wonder what should have been done.

(584) The ball, he took. *topicalization*

We have said very little regarding lexical ambiguity, although there are a few marked rules to cover some simple cases. There is one rule to distinguish an *auxiliary* from a main verb and another to separate the various uses of *that* (a complementizer, a relative pronoun, a normal pronoun, and a determiner). The first rule was discussed in chapter 7. Neither rule is particularly elegant; Milne is working on more attractive solutions to the lexical ambiguity problem.

(585) <u>Have</u> the boys take the ball! *auxiliary diagnostic*
(586) <u>Have</u> the boys taken the ball?
(587) Which boys <u>were</u> the girls taking to the ball?
(588) Which boys <u>have</u> the girls take the jars?
(589) Which boys <u>have</u> the girls taken to the ball?

(590) I know a man that was nice. *that diagnostic*
(591) I know that was nice.
(592) I know that that was nice.
(593) I know that boys are nice.
(594) I know that boy is nice.
(595) I know that he is nice.
(596) That he is nice is a fact.
(597) That that boy is nice is a fact.
(598) That that is nice is a fact.
(599) Who do you believe that was?
(600) Who do you believe that that was?
(601) Did you believe that?
(602) Did you believe that was him?
(603) Did you believe that that was him?
(604) Did you believe he did that?

We discussed pseudo-attachment briefly in chapter 4 and pseudo-gaps in chapter 8. (605), (606) and (608) illustrate the phenomena; (607) and (609) are near misses (they have only one attachment/gap).

(605) He seems nice to her. *pseudo-attachment*
(606) Put the box on the table in the kitchen.
(607) Put the box on the table. *near miss*

(608) Which boys does he want to see? *pseudo-gaps*
(609) Which boys does he want to take? *near miss*

We have been very concerned with stack allocation. (610)-(612) illustrate some borderline center-embedded sentences.[218] YAP does require one less stack cell for (610) than the others, although the reason is very complex. We don't have enough confidence in the details to trace though the entire explanation. The generalization seems to be that a complement is less acceptable in the most deeply embedded clause [Cowper76 pp. 71]. YAP finds deeply embedded complements more difficult because it is hard to distinguish them from relative clauses without storing the entire sentence on the stack.

(610) The possibility that the man who I hired is incompetent worries me.
(611) #The man who the possibility that students are dangerous frightens is nice.
(612) #The man who the possibility that I am dangerous frightens is nice.

YAP can also parse the following right branching sentences. (616) is somewhat problematic because the two *that*'s are disambiguated in the wrong order. Hence $[_{PP}$ of $t]$ is attached to *rumor*. These diagnostics are not well understood.

(613) It might seem likely that it would seem likely that he is nice.
(614) Did you hear a rumor that there was a possibility that he might say that I am nice?
(615) Did you hear a rumor that there was a possibility that he might tell me?
(616) Did you hear a rumor that there was a possibility that he might tell me of?
(617) Did you hear a rumor that there was a possibility that he might tell me of it?
(618) Did you hear a rumor that it would seem likely that he is nice?
(619) Did you hear a rumor that John wondered who said that I am nice?

218. The first two are taken from [Cowper76].

Appendix II - An Example

This appendix shows the derivation of (620). The final output (the cstructure and the fstructure) are given as (621) and (622) below.[219]

(620) Was the boy likely to sit?

```
(621) CSUBJ: [(NP-) the boy]                                    cstructure
        CHEAD: [(NP) the boy]
         CSPEC: [(DET) the]
         CHEAD: [(N) boy]
       CHEAD: [(VP) was likely to sit]
        CHEAD: [(V) was]
         CXCOMP: [(AP-) likely to sit]
          CHEAD: [(AP) likely to sit]
           CHEAD: [(A) likely]
            CXCOMP: [(VP-) to sit]
             CCOMP: [(COMP) to]
             CHEAD: [(VP) sit]
              CHEAD: [(V) sit]
(622) FSUBJ: [(NP-) the boy]                                    fstructure
        FSPEC: [(DET) the]
       FXCOMP: [(AP-) likely to sit]
        FSUBJ: [(NP-) the boy]
         FSPEC: [(DET) the]
        FXCOMP: [(VP-) to sit]
         FSUBJ: [(NP-) the boy]
          FSPEC: [(DET) the]

sentence: was the boy likely to sit?                          initial state
```

219. This source was produced by a slightly older version of YAP. Nevertheless, it should still be highly informative.

input pointer: LIKELY TO SIT ?
[(S)]
= = WALL = =
[(V) was]
[(DET) the]
[(N) boy]

YAP will undo the inversion, but first it has to parse [$_{np-}$ the boy] to trigger the marked inversion rule. This is accomplished by the rule: APPLY-DEFAULT-ATTENTION-SHIFT. No rule of higher priority can apply because up1 is looking for a subject, not a verb.

input pointer: TO SIT ?
[(S)]
[(V) was]
= = WALL = =
[(DET) the]
[(N) boy]
[(A) likely]

The determiner in down1 triggers a marked rule to predict a noun phrase: CREATE-NP-1.

input pointer: TO SIT ?
[(S)]
[(V) was]
= = WALL = =
[(NP)]
[(DET) the]
[(N) boy]
[(A) likely]

The NP is attention shifted to allow [$_{det}$ the] and [$_{n}$ boy] to attach. The next three snap-shots show the attention shift and two attachments.

```
input pointer: TO SIT ?
[(S)]
[(V) was]
[(NP)]
==WALL==
[(DET) the]
[(N) boy]
[(A) likely]
About to run: APPLY-DEFAULT-PS-ATTACHMENT

input pointer: SIT ?
[(S)]
[(V) was]
[(NP) the]
==WALL==
[(N) boy]
[(A) likely]
[(COMP) to]
About to run: APPLY-DEFAULT-PS-ATTACHMENT

input pointer: ?
[(S)]
[(V) was]
[(NP) the boy]
==WALL==
[(A) likely]
[(COMP) to]
[(V) sit]
```

Now [$_{np}$ the boy] has all of its children, but it doesn't have a mother yet. It will be returned to the lower buffer, so it can find its mother. (Slightly contrary to the discussion in chapter 3, ps-close does an attention return if up1 isn't ready to close. In this case, up1 can't close because it doesn't have a mother.

```
About to run: APPLY-DEFAULT-PS-CLOSURE

[(S)]
[(V) was]
==WALL==
[(NP) the boy]
[(A) likely]
[(COMP) to]
```

The NP in down1 triggers a marked rule (CREATE-NP--1)[220] to predict an np-, which is immediately attention shifted, leaving the machine in the following state. Then ps-attach and ps-close apply producing the next two snap-shots.

```
[(S)]
[(V) was]
[(NP-)]
==WALL==
[(NP) the boy]
[(A) likely]
[(COMP) to]
About to run: APPLY-DEFAULT-PS-ATTACHMENT

[(S)]
[(V) was]
[(NP-) the boy]
==WALL==
[(A) likely]
[(COMP) to]
[(V) sit]
About to run: APPLY-DEFAULT-PS-CLOSURE
```

220. The rule *CREATE-NP--1* predicts an *np-* whereas the rule *CREATE-NP-1* predicts an *np*.

There is nothing left to do but attention-return, hoping to trigger some other rule. In this case, it will enable auxiliary inversion. (It should have predicted an *ap-* first. This indicates a slight problem.)

```
[(S)]
[(V) was]
==WALL==
[(NP-) the boy]
[(A) likely]
[(COMP) to]
About to run: APPLY-DEFAULT-PS-CLOSURE
```

```
[(S)]
==WALL==
[(V) was]
[(NP-) the boy]
[(A) likely]
About to run: AUX-INVERSION
```

Now, ps-attach can apply.

```
[(S)]
==WALL==
[(NP-) the boy]
[(V) likely]
[(A) to]
About to run: APPLY-DEFAULT-PS-ATTACHMENT
```

Notice that $[_{np-}$ the boy] was automatically closed, removed from the buffer, after it was attached. In this older version, the closure procedure was very much like Kimball's scheme. The current scheme would not close this early; it would leave the np- in up1 and then ps-close would apply.

```
[(S) the boy]
==WALL==
[(V) was]
[(A) likely]
[(COMP) to]
About to run: PRED-DEFAULT
```

This rule selects the appropriate predicate for down1 from the lexicon.

```
[(S) the boy]
==WALL==
[(V) was]
[(A) likely]
[(COMP) to]
About to run: ATTACH-FSUBJ
```

There is a slight problem checking functional constraints with elements to the left of the head (such as subject). Consequently, they are checked by a marked rule (ATTACH-FSUBJ) which fires when up1 has a predicate and a subject. (We are currently exploring more elegant alternatives.)

```
[(S) the boy]
==WALL==
[(V) was]
[(A) likely]
[(COMP) to]
About to run: CREATE-VP-1
```

There is a marked rule to build verb phrases bottom-up. (It is probably unnecessary.) With a more symmetric default predict rule, it should be possible to eliminate most of the marked prediction rules (CREATE-...).

```
[(S) the boy]
[(VP)]
==WALL==
[(V) was]
[(A) likely]
[(COMP) to]
About to run: APPLY-DEFAULT-PS-ATTACHMENT
```

YAP finishes the parse using the same techniques.

```
[(S) the boy was]
[(VP) was]
==WALL==
[(A) likely]
[(COMP) to]
[(V) sit]
About to run: CREATE-XCOMP-1

[(S) the boy was]
[(AP-)]
==WALL==
[(A) likely]
[(COMP) to]
[(V) sit]
About to run: PRED-DEFAULT
```

```
[(S) the boy was]
[(AP-)]
==WALL==
[(A) likely]
[(COMP) to]
[(V) sit]
About to run: ATTACH-FSUBJ
```

```
[(S) the boy was]
[(AP-)]
==WALL==
[(A) likely]
[(COMP) to]
[(V) sit]
About to run: CREATE-AP-1
```

Notice that the *ap-* will close when the *ap* is attached in the next snapshot.

```
[(S) the boy was]
[(AP)]
==WALL==
[(A) likely]
[(COMP) to]
[(V) sit]
About to run: APPLY-DEFAULT-PS-ATTACHMENT
```

```
[(S) the boy was likely]
[(AP) likely]
==WALL==
[(COMP) to]
[(V) sit]
[(PUNCT) ?]
About to run: CREATE-INF-VCOMP

[(S) the boy was likely]
[(VP-)]
==WALL==
[(COMP) to]
[(V) sit]
[(PUNCT) ?]
About to run: APPLY-DEFAULT-PS-ATTACHMENT

[(S) the boy was likely to]
[(VP-) to]
==WALL==
[(V) sit]
[(PUNCT) ?]
About to run: PRED-DEFAULT

[(S) the boy was likely to]
[(VP-) to]
==WALL==
[(V) sit]
[(PUNCT) ?]
About to run: ATTACH-FSUBJ
```

```
[(S) the boy was likely to]
[(VP-) to]
==WALL==
[(V) sit]
[(PUNCT) ?]
About to run: CREATE-VP-1

[(S) the boy was likely to]
[(VP)]
==WALL==
[(V) sit]
[(PUNCT) ?]
About to run: APPLY-DEFAULT-PS-ATTACHMENT

[(S) the boy was likely to sit]
==WALL==
[(PUNCT) ?]
```

References

1. Ades, Anthony E. and Steedman, Mark, *On Word-Order*, unpublished paper, Dept of Psychology, University of Warwick, U.K., 1979.

2. Aho, Alfred V. & Ullman, Jeffrey D., *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, Inc., 1972.

3. Aho & Ullman, *Principles of Compiler Design*, Addison-Wesley Publishing Company, 1977.

4. Akamajian and Heny, *An Introduction to the Principles of Transformational Syntax*, MIT Press, 1975.

5. Akamajian, Steele and Wasow, *The Category AUX in Universal Grammar*, Linguistic Inquiry, 10:1, 1979.

6. Anderson, S. and Kiparsky (eds), *A Festschrift for Morris Halle*, Holt, Rinehart, and Winston, New York, 1973.

7. Arbib, Michael A., *Theories of Abstract Automata*, Prentice-Hall, Inc., 1969.

8. Bar-Hillel, Y. and Shamir, E., *Finite State Languages: Formal Representation and Adequacy Problems*, Bull. Res. Council of Israel, 8F, 242-256, 1960.

9. Bar-Hillel, Y., Perles, M., and Shamir, E., *On Formal Properties of Simple Phrase Structure Grammars*, reprinted in *Readings in Mathematical Psychology*, 1961.

10. Berwick, Robert., *Learning Structural Descriptions of Grammar Rules from Examples*, IJCAI79. vol. 1, 1979.

11. Berwick, R., *Computational Analogues of Constraints on Grammars*, ACL Conference Proceedings, 1980a.

12. Berwick, R., *Learning Structural Descriptions of Grammar Rules from Examples*, MIT, AI-TR 578, 1980b.

13. Bever, Thomas G., *The Cognitive Basis for Linguistic Structures*, J. R. Hayes (ed.), *Cognition and the Development of Language*, 1970.

14. Bever, T. and Langendoen, T., *A Dynamic Model of the Evolution of Language*, LI 2:433-463, 1971.

15. Bever, Thomas G., *The Psychology of Language and Structuralist Investigations of Nativism*, in Harmon (ed), 1974.

16. Bresnan, Joan, *A Realistic Transformational Grammar*, in Halle, Bresnan and Miller (eds), 1978.

17. Bresnan, Joan, *A Theory of Grammatical Representation*, class notes, MIT, 1979.

18. Bresnan, Joan, *Polyadicity: Part I of a Theory of Lexical Rules and and Representations*, unpublished paper, MIT, 1980.

19. Chomsky, Noam, *Three Models for the Description of Language*, I.R.E. Transactions on Information Theory, vol. IT-2, Proceedings of the Symposium on Information Theory, 1956.

20. Chomsky, N., *Syntactic Structures*, Mouton & Co., 1957.

21. Chomsky, N., *On Certain Formal Properties of Grammars*, Information and Control, vol 2, pp. 137-167, 1959a.

22. Chomsky, N., *A Note on Phrase Structure Grammars*, Information and Control, vol 2, pp. 393-395, 1959b.

23. Chomsky, N., *Formal Properties of Grammars*, in R. D. Luce, R. R. Bush, and E. Galanter, eds., pp. 323-418, 1963.

24. Chomsky, N., *On the Notion "Rule of Grammar"*, (1961), reprinted in Fodor and Katz (eds.), pp 119-136, 1964.

25. Chomsky, N., *A Transformational Approach to Syntax*, in Fodor and Katz, (eds.), 1964.

26. Chomsky, N., *Some Aspects of the Theory of Syntax*, MIT Press, Cambridge, MA., 1965.

27. Chomsky, N., *Remarks on Nominalization*, in Jacobs and Rosenbaum (eds), 1970.

28. Chomsky, N., *Conditions on Transformations*, in Anderson and Kiparsky (eds), 1973.

29. Chomsky, N. and Lasnik, H., *Filters and Control*, Linguistic Inquiry, 8:3, 1977.

30. Chomsky, N., *On Binding*, Linguistic Inquiry, 1980.

31. Cowper, Elizabeth A., *Constraints on Sentence Complexity: A Model for Syntactic Processing*, PhD Thesis, Brown University, 1976.

32. Crain, Stephen, and Coker, Pamela L., *A Semantic Constraint on Syntactic Parsing*, presented at the Linguistic Society of America Annual Meeting, University of California, Irvine, 1978.

33. Crain, Stephen, *Remarks on the Theory of Performance*, unpublished paper, 1979.

34. Davis, Randall, *Interactive Transfer of Expertise: Acquisition of New Inference Rules*, IJCAI, 1977.

35. DeRemer, F., *Simple LR(k) grammars*, Communications of the ACM, Volume 14, pp. 453-460, 1971.

36. Doyle, Jon, *Truth Maintenance Systems for Problem Solving*, AI-TR-419, MIT, 1978.

37. Earley, Jay, *An Efficient Context-Free Parsing Algorithm*, Unpublished Ph.D Thesis, CMU, 1968.

38. Earley, J., *An Efficient Context-Free Parsing Algorithm*, Communications of the ACM, Volume 13, Number 2, February, 1970.

39. Emonds, Joseph E., *A Transformation Approach to English Syntax*, Academic Press, Inc., 1976.

40. Fahlman, Scott E., *A System for Representing and Using Real-World Knowledge*, AI-TR-450, MIT, 1977.

41. Fodor, Janet D., *Parsing Strategies and Constraints on Transformations*, Linguistic Inquiry, 9:3, 1978

42. Fodor, J. and Frazier, L., *Is the Human Sentence Processing Mechanism an ATN?*, unpublished paper, University of Connecticut, 1980.

43. Fodor, Jerry and Katz, Jerold (eds), *The Structure of Language* Englewood CLiffs, New Jersey, Prentice-Hall, 1964.

44. Frazier, Lyn & Fodor, Janet D., *The Sausage Machine: A New Two-Stage Parsing Model*, Cognition, 6, 291-325, 1978.

45. Frazier, Lyn, *On Comprehending Sentences: Syntactic Parsing Strategies*, PhD Thesis, University of Massachusetts, Indiana University Linguistics Club, 1979.

46. Frazier, L., *Parsing and Constraints on Word Order*, in Lowenstamm (ed), 1980.

47. Freidin, Robert, *Cyclicity and the Theory of Grammar*, Linguistic Inquiry, 9:4, 1978.

48. Gazdar, Gerald, *Constituent Structures*, unpublished paper, School of Social Sciences, University of Sussex, 1979a.

49. Gazdar, G., *English as a Context-Free Language*, unpublished paper, School of Social Sciences, University of Sussex, 1979b.

50. Gazdar, G., *Unbounded Dependencies and Coordinate Structure*, unpublished paper, School of Social Sciences, University of Sussex, 1979c.

51. Greenberg, J. H., *Some Universals of Grammar with Particular Reference to the Order of Meaningful Elements*, In J. H. Greenberg (ed), *Universals of Language*, MIT Press, 1963.

52. Greibach, S. A., *Formal Languages: Origins and Directions*, IEEE, CH1471-2/79/0000-0066, 1979.

53. Grishman, R., *Implementation of the String Parser*, in Rustin, R. (ed), 1973.

54. Halle, Bresnan, and Miller (eds.), *Linguistic Theory and Psychological Reality*, MIT Press, 1978.

55. Hankamer, J., *Unacceptable Ambiguity*, Linguistic Inquiry 4, pp. 17-68, 1973.

56. Harmon, Gilbert (ed), *On Noam Chomsky: Critical Essays*, Anchor Press, Doubleday, 1974.

57. Hewitt, Carl, *Description and Theoretical Analysis (using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot*, PhD thesis, AI-TR-258, MIT, 1972.

58. Huybregts, M., *Overlapping Dependencies in Dutch*, Instituut A.W. de Groot, 1976.

59. Jackendoff, Ray, *X-Bar Syntax: A Study of Phrase Structure*, Linguistic Inquiry Monograph Two, 1977.

60. Jacobs, R. and Rosenbaum, P. (eds), *Readings in English Transformational Grammar*, Ginn. Waltham, MA., 1970.

61. Kaplan, R., *Augmented Transition Networks as Psychological Models of Sentence Comprehension*, Artificial Intelligence, 3, 77-100, 1972.

62. Kaplan, R., *Transient Processing Load in Relative Clauses*, unpublished doctoral dissertation, Harvard University, 1975.

63. Kaplan, R., *Computational Resources and Linguistic Theory*, revised version of paper presented as the Second Theoretical Issues in Natural Language Processing Conference, Urbana, 1978a.

64. Kaplan, R. and Bresnan, J., *A Formal System for Grammatical Representation*, unpublished paper, MIT, 1978b.

65. Kay, Martin, *Syntactic Processing and Functional Sentence Perspective*, TINLAP-1, 1975.

66. Kimball, John, *Seven Principles of Surface Structure Parsing in Natural Language*, Cognition 2:1, pp 15-47, 1973.

67. Kimball, J., *Predictive Analysis and Over-the-Top Parsing*, in *Syntax and Symantics IV*, Kimball editor, 1975.

68. Koster, Jan, *Locality Principles in Syntax*, Foris Publications Dordrecht, 1978.

69. Knuth, D. E., *On the Translation of Languages from Left to Right*, in *Information and Control*, vol. 8, 1965.

70. Kuno, Susumu, and Oettinger, A. G., *Multiple Path Syntactic Analyzer*, in *Information Processing*, North-Holland Publishing Co., Amsterdam, 1963.

71. Kuno, Susumu, *The Predictive Analyzer and a Path Elimination Technique*, Chapter 6 in *Readings in Automatic Language Processing*, 1966.

72. Kuno, S., *Natural Explanation for Some Syntactic Universals*, Mathematical Linguistics and Automatic Translation, Report NSF-28, The Aiken Computation Laboratory, Harvard University, 1972.

73. Kuno, S., *Constraints on Internal Clauses and Sentential Subjects*, Linguistic Inquiry, Vol. 4, No. 3, pp. 363-385., 1973.

74. Kuno, S., *The Position of Relative Clauses and Conjunctions*, Linguistic Inquiry, 1974.

75. Langendoen, T., *Finite-State Parsing of Phrase-Structure Languages and the Status of Readjustment Rules in Grammar*, Linguistic Inquiry Volume VI Number 4, Fall 1975.

76. Lasnik, H., *Remarks on Co-Reference*, Linguistic Analysis, Volume 2, Number 1, 1976.

77. Lowenstamm, Jean (ed), *University of Massachusetts Occasional Papers in Linguistics*, vol 5, 1980.

78. Luce, R. D., R. R. Bush, and E. Galanter, eds., *Handbook of Mathematical Psychology*, Volume II, John Wiley and Sons, New York, 1963.

79. Mackworth, Alan K., *Consistency in Networks of Relations*, AI Journal, February 1977.

80. Marcus, Mitchell, *A Theory of Syntactic Recognition for Natural Language*, Ph.D Thesis, MIT Press, 1979.

81. Martin, William A., *Descriptions and the Specialization of Concepts*, in *Artificial Intelligence: An MIT Perspective*, MIT Press, Winston and Brown (eds), 1979.

82. Martin, W., class notes, MIT, 1979 and 1980.

83. McAllester, David A., *The Use of Equality in Deduction and Knowledge Representation*, MIT, AI-TR-550, 1980.

84. McDonald, David D., *Steps Toward a Psycholinguistic Model of Language Production*, AI Memo 500, 1979.

85. McDonald, D., *Natural Language Production as a Process of Decision-Making Under Constraints*, Ph.D. Thesis, MIT, 1980.

86. Milne, Robert W., *Handling Lexical Ambiguity in a Deterministic Parsing Environment*, unpublished B.Sc. Thesis, MIT, 1978a.

87. Milne, R., *The Fringe of Lexical Ambiguity*, unpublished paper, MIT, 1978b.

88. Milne, R., *Using Determinism to Predict Garden Paths*, AISB 80 Conference, 1979.

89. Milne, R., *A Framework for Deterministic Parsing Using Syntax and Semantics*, DAI Working Paper 64, Department of Artificial Intelligence, University of Edinburgh, 1980.

90. Peters, S., and Ritchie, R., *On the Generative Power of Transformational Grammars*, Information Sciences 6: 49-83, 1973.

91. Petrick, Stanley R., *A Recognition Procedure for Transformational Grammars*, Unpublished Ph.D. Thesis, MIT, 1965.

92. Postal, Paul M., *Limitations of Phrase Structure Grammars*, in Fodor & Katz, eds., 1964.

93. Postal, P., *On Raising*, MIT Press, 1974.

94. Pratt, V. R., *A Linguistics Oriented Programming Language* IJCAI-3, 1973.

95. Pratt, V., *Lingol - A Progress Report*, IJCAI-4, 1975.

96. Pratt, V., *The Competence/Performance Dichotomy in Programming*, 4th ACM Symposium on Principles of Programming Languages, Los Angeles, January 1977.

97. Rich, Charles, *On the Psychological Reality of Augmented Transition Network Models of Sentence Comprehension*, unpublished paper, MIT, 1975.

98. Ross, J. R., *Constraints on Variables in Syntax*, unpublished Doctoral dissertation, MIT, 1967.

99. Rustin, Randall (ed), *Natural Language Processing*, Algorithmics Press, New York, 1973.

100. Ruzzo, Walter L., *General Context-Free Language Recognition*, unpublished PhD Thesis, University of California, Berkeley, 1978.

101. Shipman, D., *Some Ideas For Collapsing the Marcus' Parser*, unpublished term project for J. Allen, MIT, 1978.

102. Shipman, D. and Marcus, M., *Towards Minimal Data Structures for Deterministic Parsing*, IJCAI79, 1979.

103. Steele, Guy L., *Rabbit: A Compiler for Scheme (A study in Compiler Optimization)*, AI-TR-474, MIT, 1978.

104. Swartout, William, R., *A Digitalis Therapy Advisor with Explanations*, MIT/LCS/TR-176, 1977.

105. Swartout, W., *A Comparison of PARSIFAL with Augmented Transition Networks*, AI Memo 462, MIT, March, 1978.

106. Ullman, Jeffrey, *Pushdown Automata with Bounded Backtrack*, System Development Corporation, TM-738/022/00, 1965.

107. Valient, L., *General context free recognition in Less Than Cubic Time*, J. Computer and System Sciences 10, pp. 308-315, 1975.

108. VanLehn, Kurt A., *Determining the Scope of English Quantifiers*, AI-TR-483, MIT, 1978.

109. Wales, Roger and Toner, Hugh, *Intonation and Ambiguity*, in Cooper and Walker, 1976.

110. Waltz, D., *Understanding Line Drawings of Scenes with Shadows*, in Winston (ed), 1975.

111. Wanner, E. and Maratsos, M., *An ATN Approach to Comprehension*, in Halle, Bresnan and Miller (eds.), 1978.

112. Wanner, E., *The ATN and the Sausage Machine: Which One is Baloney?*, unpublished paper, Sussex University, 1979.

113. Williams, Edwin, *Passive*, unpublished paper, University of Massachusetts, Amherst, 1980.

114. Winograd, T., *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*, Project MAC-TR 84, MIT, 1971.

115. Winston, Patrick H. (ed), *The Psychology of Computer Vision*, McGraw-Hill Book Company, New York, 1975.

116. Winston, Patrick H. and Brown, Richard H. (eds), *Artificial Intelligence: An MIT Perspective*, MIT Press, 1979.

117. Woods, William, *Transition Network Grammars for Natural Language Analysis*, Communications of the ACM, Volume 13, Number 10, October, 1970.

118. Woods, W., *An Experimental Parsing System for Transition Network Grammars*, in Rustin (ed), 1973.

119. Woods, W., *Some Methodological Issues in Natural Language Understanding Research*, TINLAP-1, 1975.

120. Yngve, V. H. *A Model and an Hypothesis for Language Structure*, Proceedings of the American Philosophical Society 104, pp. 444-466, 1960.