LEARNING STRUCTURAL DESCRIPTIONS OF GRAMMAR RULES FROM EXAMPLES

by

Robert Cregar Berwick

A.B., Harvard College
(1976)

Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science

at the

Massachusetts Institute of Technology
June 1980

Signature of Author Signature redacted
Department of Electrical Engineering and Computer Science
May 9, 1980

Certified by Signature redacted
Patrick Henry Winston
Thesis Supervisor

Accepted by Signature redacted
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

LEARNING STRUCTURAL DESCRIPTIONS OF GRAMMAR RULES FROM EXAMPLES

by
Robert Cregar Berwick

Submitted to the Department of Electrical Engineering and Computer Science
on May 9, 1980 in partial fulfillment of the requirements
for the Degree of Master of Science

Abstract

A principal goal of modern linguistics is to account for the apparently rapid and uniform acquisition of syntactic knowledge, given the relatively impoverished input that evidently serves as the basis for the induction of that knowledge -- the so-called *projection problem*. At least since Chomsky, the usual response to the projection problem has been to characterize knowledge of language as a grammar, and then proceed by restricting so severely the class of grammars available for acquisition that the induction task is greatly simplified -- perhaps trivialized.

The work reported here describes an implemented LISP program that explicitly reproduces this methodological approach to acquisition in a computational setting. It asks: what constraints on a computational system are required to ensure the acquisition of syntactic knowledge, given relatively plausible restrictions on input examples (only positive data of limited complexity). The linguistic approach requires as the output of acquisition a representation of adult knowledge in the form of a grammar. In this research, an existing parser for English, Marcus' PARSIFAL, acts as the grammar. We mimic the acquisition process by fixing a stripped-down version of the PARSIFAL interpreter, thereby assuming an initial set of abilities. Only the simple pattern-action grammar rules of PARSIFAL are acquired, on the basis of induction from grammatical sentences with a degree of embedding of two or less.

To date, the accomplishments of the research are two-fold. First, from an engineering standpoint, the program succeeds admirably; starting without any grammar rules, the currently implemented acquisition version of PARSIFAL (dubbed LPARSIFAL) acquires from positive examples many of the rules in a "core grammar" of English originally written by Marcus. These include both base *phrase structure* rules that handle constituents arranged in canonical English phrase structure order and *grammar* rules such as subject-auxiliary verb inversion that handle deviations from canonical order and constituent movements. Second and more importantly, to ease the computational burden of acquisition it was found necessary to place certain constraints on grammar rule form and on rule application. The constraints on phrase structure rule form are adopted from the X-bar theory of phrase structure schemata, developed by Chomsky and Jackendoff. The constraints on rule application can be formulated as specific locality principles that govern the operation of the parser and the acquisition procedure. These LPARSIFAL constraints appear to be the computational analogues of locality restrictions proposed in several current theories of transformational grammar.

Thesis Supervisor: Patrick Henry Winston

Title: Associate Professor of Electrical Engineering and Computer Science

ACKNOWLEDGEMENTS

This thesis is about how a constrained computational representation for syntactic knowledge leads directly to a theory about how such knowledge could be acquired. Its moral is simple: a constrained representation makes the acquisition easy, eliminating countless false steps that a learner might otherwise make.

Ironically, the same moral applies to the research itself. Only in a structured setting like the MIT Artificial Intelligence Laboratory could the inductive leap to the research described here have been made. In particular I am indebted to the following people:

--to Mitch Marcus. It was Mitch's design for a constrained parser that made asking questions about the acquisition of syntactic knowledge even thinkable. But Mitch contributed much more than just the representational foundation for this research. He has served as my nominal thesis advisor; as an unending source of good ideas (many to be found on other pages of this thesis); and as a friend.

--to Patrick Winston. Advisor of this thesis, and therefore by analogy, many of the same things that Mitch has been: friend and colleague.

--to my many friends at the Artificial Intelligence Laboratory and elsewhere, who have supplied support, green Pilot pen corrections on drafts, and other things that friends are for: Candy Sidner, Marilyn Matz, Brian Smith, Beth Levin, Mike Brady, Craig Thiersch, Kurt VanLehn, and Marvin Minsky. And a special "Bureaucrats Anonymous" award to Candy for her administrative assistance.

--to the many members of the linguistic and artificial intelligence community who contributed time and conversation to this research: Noam Chomsky, Howard Lasnik, Ken Church, Ken Wexler, Steve Pinker, Bob Frieden, Aravind Joshi, Robbie Moll, and particularly Jay Keyser.

--and finally, to my parents, for providing the right initial constraints.

A Model of Syntactic Acquisition

1.1 Linguistic Constraints, Computation, and Language Acquisition

One of the important goals of modern linguistic theory is to explain why children's acquisition of language appears to be so easy. On many accounts, the "evidence" that children receive to learn language is quite impoverished and reinforcement by caretakers haphazard and ineffective. Placed against this backdrop of scattered data the strikingly uniform process of language acquisition seems doubly mysterious. Children with enormously disparate sensory environments -- normal children, deaf children of normal parents, deaf children of deaf parents, normal children of deaf parents, blind children -- all seem, at least initially, to learn the same parts of what linguists call a grammar [Newport, Gleitman, and Gleitman, 1978]. Such robust performance in the midst of raging environmental variation poses a severe challenge for any theory of syntactic acquisition based only upon general learning principles.

Modern linguistics has countered with a research strategy that neatly overcomes the learning challenge; it attempts to constrain so severely the class of possible human grammars that the language learner's burden is eased, perhaps trivialized. In Chomsky's metaphor, grammars should be "sufficiently scattered" so that children can easily select the one correctly corresponding to the language of their caretakers [Chomsky, 1965]. Such restrictions aid the learner because countless hypotheses about which possible grammar might cover the data at hand are ruled out. Consequently, many linguistic proposals for organizing grammars are motivated and evaluated with this learnability principle in mind. For example, suppose there was but a single human grammar. Such a situation would be optimal from the standpoint of language learnability: no matter how complex, the grammar could be built-in, and no learning required. More realistically, but for the same reason, many current theories of transformational grammar restrict the set of possible transformations to just a few actions plus universal constraints on their application. The business of linguistics for the past several years has been to uncover these universal principles from the "data" of grammaticality judgments, and so advance, indirectly, an explanation for language learnability.

The research reported on here attacks the acquisition problem from the standpoint of *computation*. It asks: What *computational* constraints are necessary in order to ensure the easy acquisition of a system of syntactic knowledge? The answer is provided by explicit construction of a LISP program that can acquire a variety of syntactic rules solely from grammatical example sentences. From an engineering viewpoint, the currently implemented program succeeds admirably; the majority of the rules in a "core grammar" of English have been acquired. But most importantly, it demonstrates that the goal of easy learnability is attainable if the form of the acquired grammars is tightly constrained. What makes the grammar easy to acquire is that the choices the learner must make are few. The acquisition program is limited to constructing only rules of a certain kind, built from a handful of possible

actions. The success of this approach thus confirms what is becoming a truism in artificial intelligence: having the right representation makes learning simple. What gives further support to this key finding is that the *computationally*-motivated restrictions have been independently derived via a purely linguistic-mathematical route by Culicover and Wexler [1980].

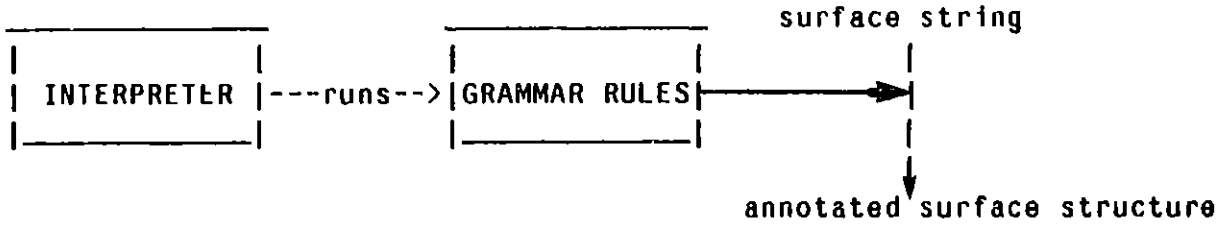
In the computational analogue to linguistics adopted here, the acquisition program must assume some "right" initial linguistic knowledge. The program developed in this research uses as its initial structure a streamlined version of PARSIFAL, Marcus' parser for English [1980]. PARSIFAL was primarily designed to handle syntactic phenomena, producing as output a modified form of the *annotated surface structures* of current transformational linguistics [Chomsky, 1973; 1976; Fiengo, 1974; 1977]. In the abstract, PARSIFAL is simply a function that takes *strings of words* to *labelled bracketings* (in the equivalent form of parse trees). *Semantic* processing is not a concern of PARSIFAL's, though it can be dealt with in a parallel fashion using the structures that PARSIFAL builds.

Computationally, PARSIFAL acts as an interpreter for grammar rules of a particularly simple pattern-action form. The patterns are triggers that determine when their associated actions should be executed; the actions are the basic operations that build the parse tree. Considered in this particularly general form, the linguistic expertise of the Marcus parser divides into two parts, a basic *interpreter* and the simple *programs* -- grammar rules -- that the interpreter executes. Figure 1.1 immediately below illustrates the division. Given this modularity, the natural way to model the acquisition of syntactic rules is to take the basic operation of the interpreter as fixed, corresponding to an initial set of abilities. Two sorts of rules are acquired:

> ★ **Context-free phrase structure rules.** (Also called *base rules.*)
> These rules determine the basic constituent order of the language, for example, that English sentences are of the form, Noun phrase-Verb Phrase, or that a Noun phrase may consist of a Determiner possibly followed by a series of Adjectives, then a Noun.
>
> ★ **Pattern-action grammar rules.**
> These correspond to syntactic rules such as subject-auxiliary verb inversion and passive.

What must be initially provided as the "basic operation" of the interpreter includes:[1]

> ★ **The major data structures and the basic control loop** of the parser; the utility programs that maintain the data structures and perform routine matching tasks;
>
> ★ **A dictionary** that can minimally classify words as either *nouns*, *verbs*, or *other*;
>
> ★ **Two skeleton phrase structure schemas**, one for Noun Phrases and one for Verb Phrases.
>
> ★ **A rudimentary well-formedness constraint** dictating that a sentence contain at least a minimal predicate-argument structure (at least a (Verb) "predicate" plus (Noun Phrase) "arguments").



Fixed	Acquired
Lexicon	Pattern-action grammar rules
Two skeleton base rules	Expanded context-free base rules
Data structures	
Rule execution loop	

Figure 1.1 - PARSIFAL's interpreter is fixed; grammar rules and expansions of base rules acquired.

Finally, it is assumed that a child, and therefore this program, uses as evidence for its hypotheses only *grammatical* example sentences, so-called *positive* data. Ruled out are presentations of *ungrammatical* sentences, followed by an indication that the example was ungrammatical, or even explicit correction of the learner's syntactic mistakes. Such pairing of ungrammatical sentences

1. A point-by-point specification of the initial state of the interpreter is laid out in Chapter 3.

followed by an indication of non-grammaticality is termed *negative data.* Although the assumption of positive-only data may well be false, on the other hand, most psycholinguistic experiments indicate that it is not [Brown and Hanlon, 1970; Newport, Gleitman, and Gleitman, 1978; summary in Culicover and Wexler, 1980]. Given our still uncertain knowledge about the linguistic evidence input to the child, the assumption of positive-only data is the strongest and safest claim we can make. If acquisition can proceed using only positive data, then it would seem completely unnecessary to move to an enrichment of the input data that is as yet unsupported by psycholinguistic evidence. Postulating negative reinforcement is dangerous on yet another ground. From formal language theory results, it is known that positive and negative examples paired with the appropriate labels "grammatical" and "ungrammatical" enable one to learn *almost any* language [see Gold, 1967]. While this result might seem fortunate, implying that negative evidence would be a boon, it also implies the existence of an *informant* who is carefully guiding the learner through some reinforcement schedule. Almost everything that is known about the early acquisition of syntax indicates that children do not ordinarily receive reinforcement of this kind. Interestingly as well, the assumption of informant presentation would implicitly place the burden of language learning on the adult, not the child, since in order to determine the next piece of data to present, the adult then must somehow know the internal state of the child's grammar.[2]

The reliance upon positive-only evidence sharply distinguishes the acquisition model of this research from most other artificial intelligence models of learning, for example, the concept-learning theory of Winston [1975]. Winston's program made essential use of negative examples as powerful evidence for hypothesis formation.[3] Perhaps the most important discovery of the research is that the limitation to positive-only evidence is not debilitating. In fact, quite the reverse is true. One can make considerable progress by thinking deeply about what sorts of constraints must take up the slack that negative evidence (supposedly) provided.

Clearly the design choices made above set aside many other difficult problems ordinarily considered to be part of language learning: How do children acquire the *meanings* of words? How do they know what sentences *mean*? This is not to belittle the importance of such questions; on the contrary, it seems obvious that the acquisition of the lexicon and the accumulation of knowledge about how linguistic utterances connect to the world should interact in significant and interesting ways with the

---

2. As pointed out in Newport, Gleitman, and Gleitman [1978] and Culicover and Wexler [1980 page xxx]. They also note that children *do* receive negative evidence for *semantic* well-formedness (e.g., the adult says "Huh?").

3. For further discussion of the mathematical theory of language learnability see Chapter 2.

acquisition of syntactic rules, further scattering the class of humanly possible grammars.[4]

But one must start somewhere. Assembling a theory that forces one to have a complete model of human cognitive development before being able to account for *any* distinctive aspect of human language learning seems hopeless. In contrast, a modular research strategy -- the usual scientific plan -- has at least some chance of success. For this reason, simplifying assumptions were made in order to focus on the problem of interest, namely, the acquisition of base phrase structure rules and grammar rules. The parallel acquisition and development of other cognitive faculties that interact with language have been largely ignored. Some comfort may be taken in this hard choice: unlike the realm of syntax, there are no outright superior candidates for a theory about "language and the world" anyway, and so no firm representational bedrock on which to ground a theory for the *acquisition* of such knowledge.

The decomposition of PARSIFAL into interpreter-plus-grammar rules can be carried one step further, a fact that allows us to push this modularity strategy further as well. To do this, one exploits the sub-systems posited by theories of generative grammar, identifying modular components in the grammar with modular components in the PARSIFAL parser. More specifically, the association between surface strings and meanings that a generative grammar provides is typically broken down into several steps. Foremost among these is a mapping between two sorts of representations, *base structures* -- the level at which predicate-argument and thematic relations such as *subject, agent, and object* are easily recovered -- and *annotated surface structures* -- an abstract representation that is much closer to the surface string's final phonological form. The *base component* is a set of context-free phrase structure rules that delimit the set of possible base structures, roughly the canonical ordering for constituents in a language; e.g., that basic English sentences are of the form Noun Phrase-Verb Phrase. The *transformational component* is a function that takes the structures so generated into annotated surface structures; transformations provide a means for dealing with deviations from the base-generated order of phrases.

It was noted above that PARSIFAL divides cleanly into two parts, an interpreter and the rules that the interpreter executes. In the original PARSIFAL, the interpreter's role was limited to the actual execution of rules and the maintenance of the resulting structure-building actions, *not* the actual decision of *when* rules should run. Almost all control of rules, in particular simple eligibility for

---

4. The current model for the acquisition of context-free phrase structure rules necessarily includes some notion about how words are acquired -- the model constructs the basic phrase structure rules via projection of the features of lexical categories. Reversing this projection, we get a preliminary wedge into the study of the lexical acquisition problem. For recent studies of lexical acquisition, the reader is referred to the work done at the University of Massachusetts: H. Goodluck and L. Solan (eds.), **Papers in the Structure and Development of Child Language**, University of Massachusetts Occasional Papers in Linguistics, volume 4, 1978; T. Roeper, J. Bing, S. Lapointe, S. Tavakolian, **A Lexical Approach to Language Acquisition**, Department of Linguistics, University of Massachusetts, Amherst, May, 1979.

execution, was encoded explicitly in the body of the rules themselves. If so, this would seem to imply that this rule control should be acquired.

However, as Marcus [1980, page 60] pointed out, by identifying the eligibility conditions for the execution of grammar rules with *phrase structure* rules, one should be able to factor this particular part of grammar rule control out of the body of the grammar rules themselves, as the figure below indicates:

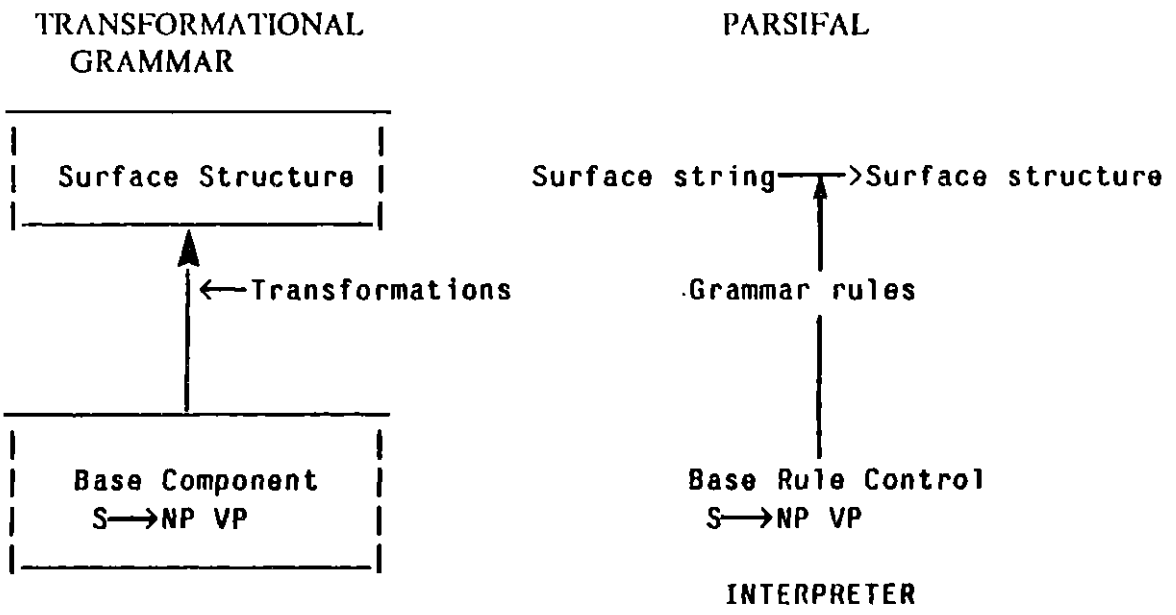


Figure 1.2 - Associating grammar rule control with the Base of a Transformational Grammar.

If the modularization is in fact practical,[5] then this control component of the parser's knowledge need not be learned as part of grammar rule acquisition itself; we can study the acquisition of grammar rules and the base phrase structure rules *separately*.

The history of the research has in fact proceeded along roughly these lines. First, a specific base component -- a full set of phrase structure rules for English -- was *fixed*. Then the acquisition of just grammar rules (minus the factored out control information) was studied.

Note, however, that by the fixing base we assume that whatever set of phrase structure rules is employed is not acquired, but pre-specified for the learner. This idealization is almost certainly false.

5. As first demonstrated by Shipman [1979] and described in Chapter 3, sections 3.2 and 3.3.

Base phrase structure rules appear to vary from language to language, and so should be set within a theory that does not demand their rigid pre-specification.

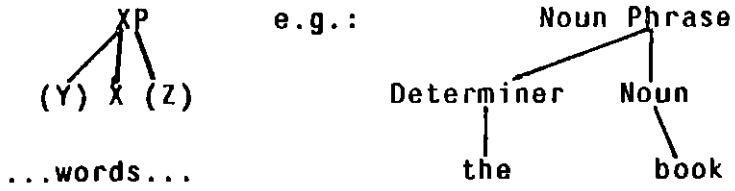
To accomodate this empirically observed variation, a phrase structure acquisition component must be added. The obvious candidate for a theory of phrase structure acquisition is the X-bar theory of phrase structure rule schemata, originally proposed by Chomsky [1970] and developed by Jackendoff [1977] and others. The X-bar theory is a good candidate for an acquisition model of phrase structure because it tightly constrains the set of (humanly) hypothesizable phrase structure rules to just a small, finite number of basic "skeletons", dubbed X-bar schemas. The original motivation for such a view came from the observation that, at a certain level of abstraction, the context-free rules for Noun Phrases and Verb Phrases look very much alike:

```
Noun Phrase⇒...Noun....
Verb Phrase⇒...Verb...
```

The theory proposes to combine the two rules into a single abstract schema, with an "X" replacing the specific terms "noun" and "verb":

```
X Phrase⇒...X...
```

More precisely, it claims that *every* possible phrase structure rule for human grammars has the skeleton form:



where the "XP" is some *category*, defined by the distinctive features of the "X" beneath it, and the flanking "Y" and "Z" are optional elements, usually "XP" categories themselves. All of these constraints were first prompted by empirical observations about English phrase structure, though they are intended to be constraints holding for all languages. That is, the X-bar theory purports to describe part of what makes a language a human language and not some arbitrary string-to-structure mapping.

The theory is restrictive because it rules out many *a priori* possible rules, e.g.,

X Phrase⇒...X XP... (e.g., Noun Phrase⇒Noun NounPhrase)[6]

Given the X-bar restrictions, an acquisition procedure can apparently easily induce the right phrase structure rules for the particular language at hand from just simple positive example sentences. For example, suppose that one already knows the basic expansion rule for English sentences, i.e., something of the form, S⇒NounPhrase VerbPhrase. Now the problem is to determine whether the proper expansion for a Verb Phrase rule is Verb Phrase⇒NounPhrase Verb or Verb Phrase⇒Verb Noun Phrase. Ignoring some presently unimportant detail,[7] a single positive example such as *Sue kissed Mitch* will serve to fix the right result. For the possible pair of rules S⇒NounPhrase Verb Phrase; VerbPhrase⇒NounPhrase Verb generates strings looking like:

NounPhrase-NounPhrase-Verb

and so cannot be fit against the given example string; however, the other possible expansion, VerbPhrase⇒Verb NounPhrase, matches the example perfectly.

Importantly, the success of the X-bar theory as an acquisition model also tells us that if the initial state of the base phrase structure system is structured enough, there need be no correlation *at all* between the complexity of the decision process to *set* the parameters of the system and the complexity of the attained state. The triggering itself can be simple, but the acquired set of phrase structure rules quite intricate.[8]

Let us return now to the other sort of syntactic knowledge to be acquired -- the grammar rules. Since grammar rules consist of patterns and actions, acquisition of these rules can be boiled down to the construction of correct patterns and actions. The central metaphor is that formulating a correct PARSIFAL grammar rule is like writing a correct computer program. Grammar rule actions correspond to the parts of a program that say *what* to do, and patterns to those parts that say *when* to

---

6. Note that there are many English phrases where two nouns occur together, e.g., *baby doll*; *garden path* -- so-called *Noun-Noun* modification. Here, one noun serves in role of an adjective (a modifier) for the other; this suggests that there is some process that modifies the labelling of a lexical of item of category N so that it can fit as an adjective. The X-bar theory as outlined in this chapter cannot deal with the complications of Noun-Noun modification, but see later in this section where some suggestions are made as to how category conversion might take place.

7. For a full acount of the same example, see the next section.

8. This point is often overlooked; consider by way of analogy the bootstrapping of a computer, where simple actions can have quite profound results.

do it. In general, writing correct programs can be extremely difficult. But in the specific case of PARSIFAL programs, if the rule actions are assumed *atomic*, the situation is not quite so grim.

The sought-after property that makes debugging a system of rules simple is *finite error detectability*: the effects of rule actions, and in particular actions that go astray, should be locally and immediately detectable as the parse tree is being built. Effects of rule actions should not propagate *very far*. This will enable an acquisition procedure to be extraordinarily simple-minded; if at some point in the parse no currently known grammar rules apply (or some known rule is in error), the procedure need consider as new candidate rules only actions applicable at the current point of failure.

To achieve the desired radius bound on rule effects, two general violation possibilities must be considered and dismissed. must be considered. First, a single grammar rule could directly affect the environment of the parse in some non-local way. This in turn could be the result of either a rule's *pattern* or its *action* having an essentially unbounded character.

To see what an unbounded pattern might look like, consider the following possible example sentence:[9]

Who did Sue tell Bob to ask Mitch to kiss?

Presumably, the learner must deduce that the underlying form of this sentence is something like:

Did Sue tell Bob to ask Mitch to kiss who?

That is, the learner must hypothesize a *who*-movement rule something like the following ("NP" = Noun Phrase; "VP" = Verb Phrase):

who NP VP NP VP NP VP ⇒ NP VP NP VP NP VP *who*

But the triggering context for this *who* movement is potentially unbounded -- there can be any number of intervening Noun Phrases and Verb Phrases between the spot where we first encounter the *who* and the place where it must be lodged.

---

9. Taken from Pinker [1979, page 265]

To write the grammar rule pattern for such an action would necessitate a potentially unbounded set of predicate tests:

If <*who* NP VP NP . . . . >
then <NP VP NP . . . *who*>

We want to rule out such a possibility by fiat. That is, we simply stipulate that no single rule pattern can make reference to unbounded context, as the *who*-movement rule did. *Rules trigger on just local context.*[10] The claim here is that this local triggering property actually characterizes the class of (learnable) human languages. In other words, one of the properties of human languages may be that they do not contain unbounded context rules, a property that makes them both easily learnable and easily parsable. If this is so, a procedure that limits itself to just bounded triggering rules will still be able to acquire a set of rules sufficient to parse English.[11]

The *who*-movement rule also illustrates how a rule *action* might be unbounded: it required placement of a token at some arbitary distance away from the point where the parser was currently building structure. This too we wish to ban outright, and for exactly the same reason. If a rule action could affect the parser's state arbitrarily "far away," then a potentially infinite chain of intervening grammar rules might apply before the effects of that rule action were discovered; if these effects were undesirable, unless there were an arbitrary backtracking facility, there would be no way to unwind all the intermediate states to uncover the guilty rule. In brief, single grammar rule actions and patterns must be local.

The possibility of chains of grammar rules leads us to the last way in which the effects of grammar rules might propagate in an unbounded fashion. Even allowing only local changes in the parser state by grammar rules, if later grammar rules use that altered state to trigger their actions, then earlier rules can still cause later ones to go astray. At first, this would seem to once again entail a great deal of laborious backtracking. Suppose that the correct parse of a sentence is represented by some correct sequence of rule applications, $R_1, R_2, \ldots, R_n$. A rule executed early in the game can theoretically cause the demise of a rule firing very much later. In this example, let us assume that R_9 is somehow detected as wrong (because it performs an incorrect action), but that it was R_2 that set up the

---

10. Note that this does not mean that people (and the proposed system) cannot acquire a rule to handle this example. It merely says that one cannot acquire *certain kinds* of rules to handle such an example. In fact, the apparently "long distance" *who* movement can be dealt with by positing a rule that iterates locally (operates "successive cyclically" as the linguists put it): *who* NP VP NP VP ...-->NP VP *who* NP VP ...--> NP VP NP VP *who* ...

11. This assumption receives strong support from work by Culicover, Hamburger, and Wexler [1975, 1977, 1980]. They prove that finite (local) error detectability does in fact suffice for the learning of a transformational grammar. See the last section of this chapter and Chapter 4 for further discussion.

environment of the interpreter that later caused R_9 to be misapplied. Now it becomes very difficult to find out what has gone wrong: one must tediously back up from R_9, undoing the action of each previous rule action step by step. At each point, all possible alternative rule actions must be tried out to see which one fixes the error at R_9-- a method that has the potential to explode combinatorially, since all possible chains of rules leading from R_2 to R_9 must be explored before the error is unwound.

The principal constraint that enforces the "no propagation" condition above follows from a stipulation Marcus originally placed on his PARSIFAL parser -- the *Determinism Hypothesis.* By *determinism* Marcus meant that all intermediate portions of the structure built in the course of a complete parse are assumed to be correct. That is, once the interpreter decides to construct a piece of the parse tree via some grammar rule action, the structure that is built is indelible. Backtracking is forbidden; one cannot undo structure-building guesses which later turn out to have been misguided. PARSIFAL *never builds any incorrect structure.*

The acquisition version of PARSIFAL -- LPARSIFAL -- adopts this claim of determinism. How does this help? Determinism aids in the assignment of blame to whatever rule directly causes an error. Although the demonstration of this claim must await Chapter 3 and a deeper presentation of the details of the acquisition procedure, a rough answer is as follows. By determinism, previously run rules are assumed to build correct structure. Suppose now that cascaded effects cannot propagate beyond a certain local radius, more particularly, that earlier rules cannot influence the mis-application of later ones (where *later* means some structural distance metric appropriately defined over phrase markers). Then if an error occurs in a parse, it *cannot* have been the fault of any distantly run grammar rule; only the current (or a "recently" fired) rule must be in error. If in turn the set of possible alternative actions is finite, we have achieved the desired goal of local and finite error detectability.

The two "if" conditions in the preceding paragraph pinpoint the additional stipulations that must be added before finite error detection becomes a real possibility. The first condition is the elimination of propagation errors; the second, the trimming of structural or triggering error possibilities.

A major claim of this research is to demonstrate that the necessary pruning can be carried out by:

> Reducing the number of grammar rule actions to just a few atomic operations.
>
> Storing as grammar rule patterns only the current local context of the state of the interpreter.
>
> Stipulating that rule actions can modify only extremely local context.

Once all this is accomplished, the acquisition procedure itself is simple. Suppose that the program reaches a point in its parse where none of its known rules apply. Given determinism and grammar rules built from just a few (and, as it turns out, mutually exclusive) actions, to build a new rule that does work one need only try each of the actions in turn at the point where failure was first detected and save the one that succeeds. Finally, the current (local) state of the interpreter can be stored as the triggering pattern for the new rule. This astonishingly simple procedure forms the heart of the acquisition program. The stipulations that permit such a simple plan to succeed -- the determinism and locality constraints -- appear to be the *computational analogues* of many currently proposed restrictions on transformational grammars.

1.2 Two Simple Scenarios

The best way to grasp how the acquisition program works in detail is to put it through its paces. A typical acquisition session consists of the presentation of a series of example sentences. The program attempts to parse each one; every time it gets stuck, it tries to construct a new rule. In this section, two scenarios will be presented that illustrate different aspects of syntactic acquisition, corresponding to the distinct *base* and *grammar rule* modules.

The first example shows how the X-bar theory can be used in a data-driven fashion to constrain the task of inducing a new phrase structure rule for Verb Phrases. A short epilogue to this story indicates how the same theory can be turned around in a predictive mode as a possible theory for the acquisition of new lexical items and new phrase structure categories.

The second scenario demonstrates how, given the base rules and grammar rules to handle simple declarative sentences, a rule of Subject-Auxiliary inversion for some English questions can be built.

Both scenarios omit irrelevant details. In particular, in an effort to concentrate on just phrase structure acquisition, the X-bar example ignores almost all of the specific operations of the actual parser.

The section then closes with a brief, pseudo-algorithmic listing of the entire acquisition procedure.

1.2.1 Acquiring a Verb Phrase Rule

To see exactly how the X-bar constraints can simplify the phrase structure induction task, suppose that the procedure has already acquired the phrase structure rule for English sentences, i.e., it knows the expansion,

Sentence⇒Noun Phrase Verb Phrase

Knowing this rule also means knowing when the rule applies, namely, that it is triggered (in English) by the presence of a Noun Phrase-Verb cluster in the input stream. Suppose further that a Noun Phrase rule to handle proper names is known:

Noun Phrase⇒Proper Name

However, the rest of the expansion for an English Verb phrase is not yet known:

Verb Phrase⇒???

The X-bar theory cuts through the maze of possible expansions for the right-hand side of this rule because it stipulates that all phrase structure rules are of the form,

"X"Phrase⇒(Y) X (Z)

where the "X" stands for an *obligatory*, already-known category, such as *Noun* or *Verb*, and "Y" and "Z" are *optionally* filled slots for other categories different from X.[12] If we replace the "X" with the category "Verb", and further assume that Noun Phrases ("NounPs") are the only other known category type, the X-bar theory tells us that the only possible configurations for a Verb Phrase rule are:

VerbP⇒NounP Verb
VerbP⇒Verb NounP
VerbP⇒NounP Verb NounP

Note that if the X-bar restrictions were not at our disposal, nothing could rule out bizarre hypotheses such as:

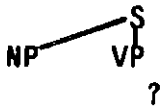
VerbP⇒Verb VerbP NounP
VerbP⇒NounP

Finally, suppose that the acquisition procedure can classify basic word tokens as *nouns*, *verbs*, or *other*.

Now the procedure is given the sentence *Sue kissed Mitch*, and commences its parse. Ignoring details about how tokens of the input stream come to the attention of the parser, note that the first token of the input, *Sue*, meets the known conditions for a Noun Phrase, and that the second token, *kissed*, is a Verb.

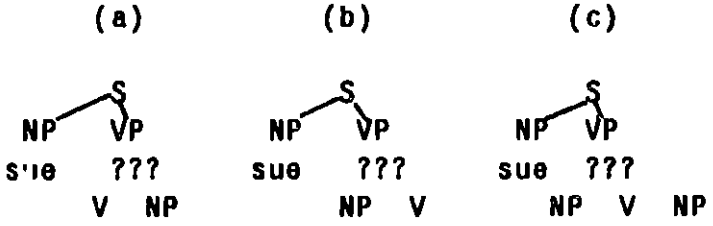
12. This is not quite accurate, but will do for this simple example.

This sparks a prediction that the Sentence phrase structure rule has been entered, and the corresponding skeleton tree is formed:[13]



Input string: Sue kissed Mitch.

Since *sue* meets all conditions for Noun Phrase-hood, it is *attached* to the NP node of this tree. (This action is actually performed by a grammar rule.) This leaves the parser in the following state, still undecided as to which of the three possible VP expansions to take:

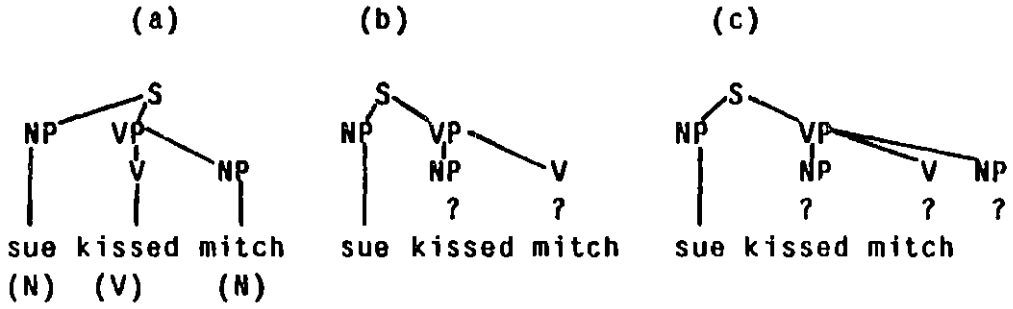


Input string: kissed Mitch.

But clearly the correct route can be quickly deduced. The next item in the input stream, *kissed*, is a verb, hence not a possible Noun Phrase. This fact rules out possibilities (b) and (c) above, but permits (a), since in (a) *kissed* could be attached as the Verb portion of the tree:

---

13. That is, the Sentence schema is triggered in a *data-driven* fashion by the presence of the Noun Phrase and the (tensed) verb. This is a modification of the original PARSIFAL parser. In Marcus' scheme, the main sentence was created predictively, that is, the parser always assumed that it was handling at least a sentence, and so automatically entered the S schema upon detecting the very first word of the input stream. However, since Marcus required data-driven Sentence creating rules anyway -- to handle embedded sentences -- it is natural to extend the data-driven triggering of Sentences to main Sentences as well. See Chapter Three for additional discussion.



The conclusion: only one Verb Phrase expansion can successfully apply to the given string, Verb Phrase⇒ Verb(V) Noun Phrase(NP). This is exactly the right result for English.[14]

In the example above, the X-bar constraints were used in a purely *data-driven* fashion; that is, the distinguishing features of the words in the input stream were used to force a particular category ordering (phrase structure tree). But the theory can also be employed in the opposite direction -- top-down or *predictively* -- as a way to classify words whose features are currently unknown.

This dual use of the X-bar theory is still under exploration. However, since the basic idea is so promising, a brief example will be given.

If we already know that we have hold of a phrase of type "X", then an unknown word in the input stream that is in the right spot to fill the "X" position of the expansion must have all the features of the type "X" category. For example, given the string, *the tove kissed Mitch*, if the article *the* is known to unambiguously initiate Noun Phrases, then the schema for an NP will insist that some "N" be found immediately. Only *tove* will do the trick.[15] The new lexical item is therefore labelled with all the properties of NP-hood (whatever these might be).

This remarkably simple insight into how new words might be categorized has all kinds of profound implications, only some of which can be covered in this report. First, it is an embryonic theory of lexical ambiguity: items of a category "X"-- say, Nouns -- can be converted into items of another category "Y" -- say, Verbs-- simply by the addition of surrounding context items drawn from category "Y", those items being simple morphemes, such as affixes, or full words. For example, English nouns can be "verbed" by adding the *ed* morphology typical of verbs; likewise, verbs can be forced into the NP category by mere appearance in an unambiguously NP context: *the broken bottle.*

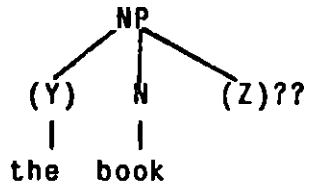
---

14. Note that if the language were basically Subject-Object-Verb, the appropriate input example *Sue Mitch kissed* would also serve to fix the right expansion, namely, VP-->NP V.

15. Attaching *tove* as some sort of adjective, and waiting for the main Noun can only lead to disaster, since no main verb remains to fit the S-->NP VP expansion.

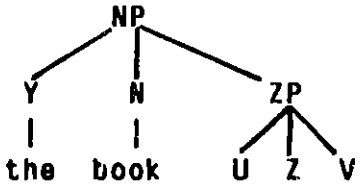
In this light, there is no "mystery" about the ubiquity of lexical ambiguity; *all* items that can appear in main "X" positions are categorically labile. Work is underway exploring this position.

Second, the predictive-mode X-bar theory suggests how new phrase structure categories are formed. Consider by way of example the Noun Phrase fragment, *the book behind the table.* If this string is somehow known to be a Noun Phrase (perhaps by appearance in proper argument position with a predicate, e.g., *NP is red*), then the unknown token *behind* poses a problem for the NP X-bar schema. Recall that by the conventions of the X-bar schema the items flanking the "N" must themselves be "XP" categories.
Assume now that *the book* has already been fit into an Noun Phrase schema:



Input string: behind the table

Let us make the further (reasonable) assumption that *behind* comes labelled as neither Noun nor Verb. Where is this item to go? By the conventions of the X-bar theory, if *behind* is to fill the "Z" slot of the Noun Phrase schema above, it must actually be part of a whole new "ZP" schema:

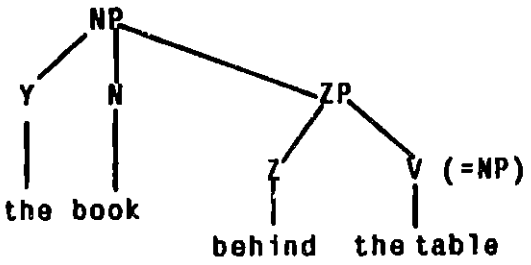


???

behind

The new "ZP" category cannot itself be an NP (a Noun Phrase), for this would violate the restrictions on all "XP" rules that just a single "X" -- not another XP -- can be the immediate descendant of an XP. What are the remaining choices for the new ZP rule? *Behind* might be fit as the "U" portion, but that would leave only the (supposedly known) NP, *the table*, as the main "Z" filler. This would be fatal, because then, by definition, the new "Z" category would have all properties of an NP, and we would have produced an expansion of the form, NP-->N NP -- exactly the form that is taboo. The

only recourse is to set *behind* as the basis for *defining* a new "ZP" phrase structure category, inserting the NP *the table* as the "V" portion of the new phrase structure rule:



And of course this is exactly the right result; the procedure has acquired its own notion of a *Prepositional Phrase.* (This part of the acquisition system is still experimental; the predictive operating mode has not yet been implemented as part of the currently running LISP program.)[16]

Even in this brief outline, the contribution of the X-bar theory to a theory of language acquisition should be apparent: its constraints provide a key to many previously inaccessible problems. The induction of context free phrase structure rules is known to be extremely difficult *in general*; the X-bar theory cracks this problem by showing that we need not consider the induction for cases "in general," but rather only for highly specific situations where particular assumptions about hypothesizable phrase structure configurations can be made.

16. There are several other issues to settle; for example, How does one decide whether to operate predictively or not? Is the distinction between predictive vs. data-driven processing reflected in the structure of the lexicon, e.g., in the difference between lexical items such as nouns and verbs (so-called *open class* items because there are a potentially unbounded number of them) vs. function words such as *that* or prepositions (*closed class* items)?

1.2.2 Acquiring a Subject-Auxiliary Verb Inversion Rule

So much for the acquisition of phrase structure rules. What about the second part of LPARSIFAL's knowledge of syntax, the grammar rules? These rules provide a means for dealing with deviations from the canonical base-generated order of phrases. Consider one such deviation: in certain English questions, what is usually called the *auxiliary verb* (or *helping verb* in some elementary school dialects) appears before the Subject of the sentence:

Did Sue kiss Mitch?

The scenario below shows how the acquisition procedure can acquire a single grammar rule to deal with this situation. It opens at a point where the only grammar rules known to the program are those for parsing simple declarative sentences -- *Sue kissed Mitch* or *Sue did kiss Mitch* can be handled. Now assume that the program receives the sentence, *Did Sue kiss Mitch?*, a question that can be answered with a yes or a no.

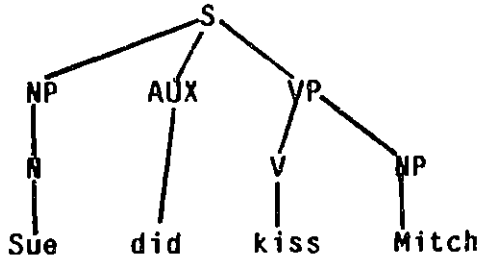
To understand how the acquisition works, some of the inner workings of PARSIFAL must first be sketched; after this, the narrative returns to the main business at hand. (The reader already familiar with PARSIFAL can advance directly to page 28.)

PARSIFAL is fashioned around two major data structures, motivated by the theoretical goal of determinism (described above) and the more practical goal of building a parsed representation of the input sentence. As a parser, PARSIFAL outputs syntactic trees closely resembling the annotated surface structures of current transformational theory. [17] A tree structure for *Sue did kiss Mitch* might look like that in Figure 1.3 (a) on the next page. Part (b) of this figure shows a snapshot of this same tree as represented in PARSIFAL by a stack of constituent nodes. The S(entence) node (labelled "S20") is on the top of the stack;[18] underneath it lie the already completed Noun Phrase (NP) and Auxiliary Verb (AUX) nodes. As pictured here, the Sentence is not yet completely parsed -- in PARSIFAL terminology, it is still *active* -- and the same is true of the Verb Phrase node VP22 (the Noun Phrase object *Mitch* has not yet been attached to the Verb Phrase). It is because there are *two* as-yet-unfinished constituents that we need a *stack* of active nodes. VP22, at the bottom of the stack, has become for the moment the focus of the parser's efforts. That is, the Verb Phrase node is the current active node (denoted C), and grammar rules that fire will attempt to build structure under

17. The distinction being that PARSIFAL trees are labelled with additional features and encode the effects of certain local, so-called "minor movement" rules directly.

18. Marcus inverted the usual convention that the *top* element of a stack refers to the first accessible item; in PARSIFAL, it is the *bottom* item on the stack that is the locus for a push or a pop. This was done so as to comport with the graphic convention that a parse tree is built *top-down*, with the accessible frontier of the tree at the bottom.

this Verb Phrase node. Notice that the verb node (*kiss*) underneath the verb phrase has already been attached, but not, as mentioned, the Noun Phrase *Mitch*. When the Verb Phrase node is complete, the parser will attach it to its place in the S node above, removing it from the stack of active nodes, and assigning the S node the status of *current* active node.



(a) Surface structure tree

```
--------------------
|  S20             |
|     NP : (sue)   |  <---Top of stack
|    AUX : (did)   |
|                  |
--------------------
|  VP22            |
|   VERB : (kiss)  |  <---Bottom of stack
--------------------
```

(b) Emerging node stack representation of tree.

Figure 1.3 - Surface structure tree and snapshot of PARSIFAL stack representation.

The second major data structure of PARSIFAL, reflecting the concerns of the Determinism Hypothesis, is a small, three-cell constituent buffer. It is the buffer that holds incoming words from the sentence string, or phrases whose grammatical function has not yet been completely determined. As Figure 1.4 below illustrates, each cell in the buffer can hold a single word or several, if these words all lie under a single node (such as a noun phrase node).

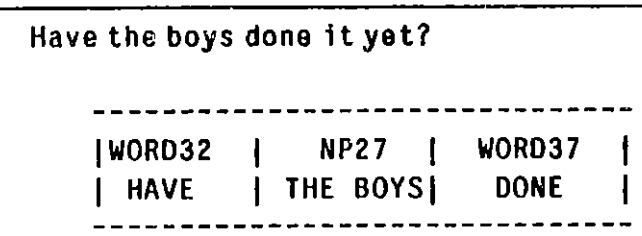


Figure 1.4 - PARSIFAL's buffer holds words or constituents.

The buffer aids the cause of determinism via its ability to hold upcoming words or phrases that are not yet completely parsed. PARSIFAL delays deciding about what syntactic structure should be built -- that is, what nodes or tokens to attach to what other nodes in the active node stack -- until it has had the opportunity to use (if necessary) the local context information in the buffer. The determinism hypothesis claims that by postponing structure-building decisions in this fashion, no choices will ever have to be undone. The necessity for such a look-ahead facility in a parser that operates left-to-right can be seen from a cursory examination of pairs of sentences like those below, from Marcus [1980]:[19]

(a) Have the boys who missed the exam take the makeup today.

(b) $[_S[_{VP}$ Have $[_S$ the boys who ... take the makeup today]]]

(a) Have the boys who missed the exam taken the makeup today?

(b) $[_S [_{AUX}$ Have$][_{NP}$ the boys who ...$][_{VP}$ taken the exam today]]

To quote Marcus,

> It is impossible to distinguish between this pair of sentences before examining the morphology of the verb following the [noun phrase] "the boys". These sentences can be distinguished, however, if the parser has a large enough "window" on the clause to see this verb; if the verb ends in "en" (in the simple case presented here), then the clause is a yes/no question, otherwise it is an imperative. Thus, if a parser is to be deterministic, it must have some constrained facility for look-ahead.
> [1980, page 17]

The typical action of grammar rules is to remove items from the first cell of the buffer and attach

19. The Marcus device might be classified as roughly an LR(k) parser, that is, a left-to-right, bottom-up parser that utilizes a k-token look-ahead. In the usual definition of LR(k) parsers, the k-token look-ahead refers to actual lexical items, e.g., individual words. In the Marcus scheme, the notion "token" is broadened to include complete constituents as well.

them to the current active node -- the lowest item on the active node stack. In addition, if the grammatical role of a constituent is undetermined, rules can insert such items back into the buffer. Recall as well that grammar rules fire if and only if their associated patterns are true of the current environment of the machine. More specifically, a pattern is some combination of features predicated of the items in the buffer and the current active node. Features are typically grammatical descriptive elements recovered from the lexical retrieval of items in the sentence string or added by the parser's own actions; they include morphological items like number (plural/singular); tense (past/present/future); and verb subcategorization (transitive/intransitive). Figure 1.5 below displays a prototypical pattern and action for operating on an auxiliary verb, first in English, and then in an abbreviated notation (close to the form actually used by the PARSIFAL interpreter).

```
        Rule Pattern
item examined              feature to match against
current active node          Major sentence
1st buffer cell                Auxiliary verb
2nd buffer cell                none
3rd buffer cell                none
        Rule action
Attach first item in buffer to current active node.
```

(a) Pattern and action for an auxiliary verb parsing rule.

```
Pattern: Current active node (C)          buffer
                                          1st   2nd   3rd
   [**c; * is S MAJOR]                   [=AUX][   ][   ]-->
Action: attach 1st to c.
```

(b) Abbreviated form.

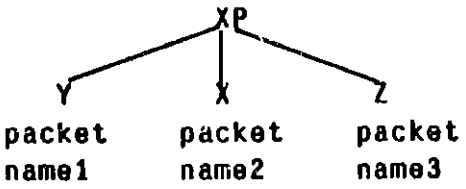
Figure 1.5 - A typical rule pattern and its abbreviated form.

In simple English, the rule in Figure 1.5 says that *if* the first item in the buffer has the feature *auxiliary* and the current active node is an *S*(entence), *then attach* the first item in the buffer to the current active node. (The items currently in cells 2 and 3, if any, will automatically slide over to take up the slack space.) Importantly, the limited horizon of pattern matching supports the locality goal discussed above: only immediate context (buffer plus current active node) is examined to determine whether a given grammar rule should apply.

Finally, functionally related grammar rules are grouped into *packets*. It is the packet system that implements what was casually referred to previously as "eligibility conditions" for grammar rules:

grouping rules into packets provides a way of controlling whether whole sets of rules should be made available for matching against the buffer items. For example, all the rules that parse Noun Phrases can be clumped together, and unless this Noun Phrase packet is *activated*, the parser will not even attempt to match the rules in this packet against the buffer. Naturally, packets must at times be *deactivated*. As described briefly in the previous section, this on-and-off switching of sets of grammar rules is carried out by associating each packet with one or more of the components of the base (phrase structure) rules of a generative grammar.

In somewhat more detail, the actual association of packets with base rules is carried out by literally pairing a packet name with each part of an X-bar schema:



The X-bar tree imposes an ordering on the activation/deactivation of packets. After a schema is entered, packets are activated left-to-right as specified by the tree. In the simplified skeleton tree above, this would mean that if the XP schema were entered, the packet associated with the "Y" portion of the tree -- presumably, a packet containing grammar rules to parse "Y" constituents -- would be activated. After Y was built (or skipped, since, it may be recalled, the "Y" component of the tree is optional), packet1 would be de-activated and the next packet in line, packet2, activated.

What happens when we come to the end of a schema? There are three possibilities. First, we might have completely built the "XP" sub-tree, and also have determined where it should be attached in the main parse tree under construction. If so, all is well; the parse just continues. [20] On the other hand, the "XP" might have been completely built, but its attachment to any higher nodes as yet undecided. If this condition holds, the parser simply drops the completely built XP node into the first cell of the buffer, shoving the current occupants of the first through third buffer cells (if any) to the right by one position. The parse then proceeds. [21] Finally, it might happen that we run off the end of the phrase structure schema without completing the XP construction. This spells trouble; we have run out of rules to build the constituent, yet the constituent is not yet in its final form. If the parser were not an acquisition model, this incident would suggest an error somewhere, either an ungrammatical sentence

---

20. In more detail, completion of the "XP" will pop "XP" as the top node in the "current active node stack" and uncover the next node down as the new current active node, along with any associated active packets.
21. Here too the active node stack is popped.

or some flaw in the packet construction or grammar rules. But in an acquisition mode, it would seem more plausible that this situation would arise from an incomplete set of rules than from an ungrammatical sentence. In short, it is the learner who shoulders the responsibility for errors, not the adults supplying the example sentences. Since this research is about acquisition, the current procedure adopts the second option: if the end of an X-schema is encountered, it plunges ahead and drops whatever part of the "XP" structure it has built back into the buffer.

A specific example will illustrate. Suppose that the phrase structure rule for starting sentences looks like:

```
S⟶      NP                  VP
Parse-S Parse-SubjectNP     Parse-VP
```

This base rule is to be interpreted as follows. If the Sentence phrase structure rule is entered (recall that it is usually triggered in a data-driven fashion by the elements NounPhrase-Verb), then the packet associated with the S(entence), Parse-S, is automatically activated; the current active node -- the locus of the parser's construction efforts-- is also set to be the S. In addition, a pointer is set to the NP component of the base rule, activating the associated packet Parse-SubjectNP. Now several things can happen. If a rule in this packet detects in a data-driven way the presence of an Noun Phrase -- by noting, say, the existence of an unambiguous flag for a Noun Phrase such as the determiner *the* in *the book* -- then additional action must be taken; since the NP is itself an X-bar schema, the packet system must be invoked a second time. To do this, the program maintains a *stack* of pointers to active nodes and active packets. In the case at hand, it simply pushes the current active node and its associated packets, S--(Parse-S; Parse-SubjectNP) o nto a stack and sets the new current active node-packet cluster to NP--(Parse-NP Parse-Y), where "Parse-Y" refers to whatever packet is associated with the left-most production of the X-bar expansion for NP's, ie., NP-->Y NP Z.

After all possible rules in the NP-associated packets have had a chance to match and run, a Noun Phrase will supposedly have been successfully built. The active node stack will be popped, uncovering the "S" once again as the current active node, and the packet system for S as the active packet; the Noun Phrase will be dropped into the buffer. At this point, if one assumes a "mature" parser, a grammar rule in the S-associated packet should fire and attach the Noun Phrase just dropped to the S, as desired. Then, the pointer for the S system will be stepped to the next part of the base rule, activating the Verb Phrase packet.

With the the Verb Phrase packet activated, the process repeats itself. If a Verb Phrase truly follows the Noun Phrase, it will ultimately be completely built, and finally attached to the "S". As usual, the pointer to the S schema will be incremented, but this time, the step takes the pointer past the end of the S schema. If the sentence is also at an end, well and good.

(If it is not, the situation is more complex; the S just built may be part of an embedded sentence. This particular outcome will not be covered in this Chapter.)

With this explanation in mind, we can now return to the acquisition of the yes-no question rule. Given the sentence *Did Sue kiss Mitch?* the parser immediately bogs down. Why? Recall first that by assumption the parser's grammar rules include all and only the rules to parse simple declarative sentences. But the current state of the first buffer cell for the example sentence is something like:

```
1st: [Did,  Auxverb.....]
2nd: [Sue,  NP....       ]
```

As can be seen from the features in the second buffer cell above, this analysis presupposes that the Noun Phrase *sue* has already been parsed and automatically returned to the second buffer cell labelled as such ("NP"). This is an automatic feature of the original PARSIFAL parser, but the ability of the parser to to "shift its attention" in this way in order to pre-package Noun Phrases requires some additional bookkeeping that will not be described here; for further discussion, see Chapter 3.

Continuing, because the S(entence) phrase structure schema has presumably been triggered,[22] the following packets are active as well:

```
   S⟶      NP VP
   ↑       ↑
Parse-S  Parse-SubjectNP
```

What must grammar rules in either of these packets look like? First, observe that the Parse-S packet will not contain any rules. This is because the Parse-S grammar rules must (by definition) be those that link S(entences) to other S(entences) -- that is, they are rules that handle embedded sentences. Since only declarative sentences have been assumed to be currently parsable, there are no such rules

22. Otherwise, the current active node could not be "S". How is this actually done? The *tense* of *did* and the NP *Sue* are enough to do the trick.

currently in the procedure's database.[23] What about rules in the Parse-SubjectNP packet? Their patterns must match on features typical of the "leading edges" of declarative sentences. That is, they must have patterns something like:

```
Buffer:
        1st:  [NP     ] (as in Sue kissed...)
        2nd:  [some combination of features]
        3rd:  [some combination of features]

        current
        active
        node: [S(entence)]
```

The program will strive in vain to match rules with triggering patterns like [NP][verb] against a buffer that has the features [auxverb, verb][NP]. The parser is stuck; a new grammar rule must be hypothesized. As we know now, specifying a rule means providing a workable action, a pattern, and an associated packet.

The last two are easy. We assume that the place to build a new rule is the place where the parser has gotten mired, and so the packet to put the new rule into is the one that was active at the time of failure. In this example, an NP-attaching packet, *Parse-SubjectNP*, is currently active. Note that being able to pinpoint errors in this way is dependent upon two properties: no backtracking, adopted in PARSIFAL; and no error propagation, a *claim* of this thesis. The right pattern to save must be one that ensures the firing of the new rule if the situation that caused the parser to get stuck ever arises again. What could be simpler than to literally store the features of the buffer and current active node as the new rule pattern?[24] This will force any future duplication of the error-causing environment to

---

23. As an example, consider *For John to kiss Sue would be unusual.* This (complex) sentence consists of *two* smaller sentences:

Sentence 1: NP--would be unusual

Sentence 2: John-kiss-Sue

The complete sentence is composed out of these two smaller core element S's by first converting the second S into a Noun Phrase via an extension of the same X-bar conversion device described in the previous section; we add the context element *for*:

for-S⇒Noun Phrase

and then using the resulting NP to fill the slot demanded by the first sentence. It is the *for*-S sort of rule that we might expect to find in the Parse-S packet. However, since by assumption the only rules the program knows about are those that can parse simple declarative sentences, no such S-composing rules exist. The Parse-S packet is empty.

24. The complete implementation additionally stores the features of the *cyclic node* (Noun Phrase or Sentence node, if any) immediately above the current active node. In the case at hand, there are no nodes above the current S node, so the pattern for the cyclic node is set to *nil*.

run the right extricating rule.[25] Having specified the packet and the pattern, one is left with discovering the right action. It is time to raise the curtain on the possible choices for rule actions. They are:

1. Attach the first item in the buffer to the current active node.

2. Switch (interchange) the first and second items in the buffer.

3. Insert one of a small number of specific lexical items into the first cell of the buffer. (e.g., *of, to*)

4. Drop a special dummy node, called a *trace*, into the first cell of the buffer.

Note that the arguments to actions 1, 2, and 4 -- which elements the action operates on and where -- are fixed. When performing a switch the program does not need to learn which buffer elements are interchanged. Only in the case of *insert ⟨lexical item⟩* must the proper argument (the lexical item inserted) be learned.[26] To finish off the new grammar rule, the acquisition program must select one of these four actions. It does so in the most obvious way, trying each of them in turn until one succeeds. Such a procedure entails a notion of *success* and *failure* for each action:

Attach is subject to type checking via the X-bar theory. That is, a node or word must be attached to a node of a compatible category, in a sense defined by the conventions of the X-bar account; Nouns can fit under Noun Phrase nodes, Verbs under Verb nodes, Prepositions cannot be attached under noun nodes, and so forth.

Switch succeeds if interchanging the items in the first and second cells enables an already known rule to run or an *attach* to succeed (the *attach* being subject to the same X-bar checks as any *attach*). Likewise, *insert ⟨lexical item⟩* must result in the immediate triggering of a known rule or a valid *attach*. The insistence that either of these actions must be followed by the execution of a known rule or an *attach* is intended to ensure that

---

25. The pattern **[did, auxverb ...][sue, NP, ...]** will also be generalized by future instances of subject-auxiliary verb inversion; see footnote 26 below.

26. Nothing more particular will be said in this introductory scenario about the insert and drop actions. However, a word about traces. They are intended to function as in Chomsky's theory of annotated surface structure [Chomsky, 1977; Fiengo, 1974, 1977], and mark the place *from which* certain constituents, like noun phrases, have been moved. Their binding -- e.g., which NP they are a placemarker for -- is determined after the initial parsing by a component that is exogenous to the syntactic theory presented here. For the most part it appears that the bindings can be calculated quite simply from syntactic and lexical considerations [Chomsky, **On Binding**, 1980]. Consequently, interfacing such a component to the parser poses no special computational problems; the details do not bear on the acquisition program implemented here and will not be further discussed in this introductory Chapter.

the acquisition procedure is not applied recursively. In addition, *switch* obeys a locality constraint: roughly, it can only exchange nodes adjacent within a local domain.[27] For *insert*, test items are chosen in succession from a small list of insertable elements, adopting the same type checking as for *attach*.

Drop (trace) works if the annotated surface structure that would be built can have its trace bindings correctly determined by a set of semantic interpretation rules (not specified in this research). Note that since trace interpretation operates "at a distance", this last rule has an unbounded character that might pose some difficulties for a purely local rule refinement scheme. However, the apparent trouble evaporates upon careful analysis. As will be shown in Chapter 3, the effect of ordering *drop trace* after the other rule actions provides an appropriate reign on the non-local behavior of traces.

We now return to the rule bottleneck in the example sentence. Considering each potential action in turn, the acquisition program first tries *attach*. But *attach* must fail, for the auxiliary verb *did* cannot be placed under the current noun phrase node the parser is building; auxiliary verbs are not part of noun phrase constituents. The next choice, *switch*, is more fortunate. *Did* is exchanged for *Sue* and the buffer now looks like:

```
1st:    [Sue NP, Noun...]
2nd:    [did, Auxverb...]
```

Now the machine is in precisely the state of parsing a simple declarative sentence. By assumption, the parser can handle this kind of sentence; consequently, the same rule that attaches *Sue* as the Subject Noun Phrase in the sentence *Sue did kiss Mitch* must now be able to successfully match against the new buffer state. The *switch* has succeeded.

---

27. More precisely, only nodes that *c-command* each other in a cyclic environment. For the exact formulation of this constraint, see Chapter 3, section 4.

Celebrating its success, the program saves the new rule:

```
RULE-<name> in packet Parse-SubjectNP
PATTERN:
   Current
   active        [S(entence)]
   node

   1ST:          [did...]
   2ND:          [Sue...]
   3RD:          [ empty]

ACTION:       switch

(The <name> of the new rule is supplied by some
external agency for mnemonic purposes.)
```

Finally, the program must add a new feature to the current active node (here, the S node): the name of the rule just applied. The reader may have noticed that such an encoding is necessary. Otherwise the structure built for switched sentence would be identical to that for the corresponding simple declarative; there would be no way to determine that the sentence was originally a yes-no question. Labelling the S node with the name of the *switch* rule records the event of auxiliary inversion so that this distinction can be maintained for later use.

Note also that the newly stored rule is as specific as possible: its triggering pattern incorporates every possible feature associated with the tokens *did* and *Sue*. Literally speaking, the newly acquired rule can only handle an exact repetition of the sample sentence. The way out of this bind is for the procedure to encounter additional positive examples; later instances of Subject-Auxiliary verb inversion can be used to relax the over-specificity. Briefly, if at a later time another auxiliary-verb inversion sentence is presented, say, *Has the dog bitten Mitch?*, the acquisition procedure will eventually find itself constructing a new *switch* rule to be placed in the packet Parse-SubjectNP. But before actually storing the new rule, it will first check to see whether any other rules in the packet perform the same action; if so, it will merge the rules with common actions by intersecting their patterns into a generalized form. In the *switch* case, the acquisition procedure will indeed discover another rule in the Parse-SubjectNP packet that performs a *switch*.

Intersecting the rule pattern features of the old rule and the corresponding rule produces a generalized *switch*:

```
          old rule                  new rule                merged rule
1st:   [did, auxverb...    ]+[has, auxverb...    ]-->[auxverb...        ]
2nd:   [sue, NP...         ]+[the dog, NP...     ]-->[NP...             ]
3rd:   [something          ]+[something else     ]-->[anything          ]
current
active [      sentence     ]+[      sentence     ]-->[      sentence    ]
node
  Action: switch
```

The resulting merged pattern is exactly that of the actual yes-no question rule as implemented by Marcus [1980].[28]

28. Actually, the above account has omits several key elements of the generalization process. First, all the patterns discussed in this introductory chapter have omitted reference to an additional component of rule patterns, the *cyclic node above the current active node* -- i.e., an immediately higher S(entence) or Noun Phrase, if any. This extra node can play an important role in certain rule patterns. In the case at hand, it distinguishes between main (root) sentences, where there is no "higher" cyclic node (e.g., simple sentences like *Sue did kiss Mitch*; the cyclic node above the main S would thus be NIL), and embedded sentences, where there is a higher cyclic node (usually, a Noun Phrase or another sentence), as in *I wonder who kissed Mitch?*. The distinction between root and non-root sentences is important for auxiliary verb inversion because the inversion rule does not generalize to embedded sentences: **I wonder did Sue kiss Mitch.* Since no such sentence will ever be among the positive examples the program receives, given an inversion rule for main Sentences with a pattern stipulating an empty cyclic node, the program will never change its feature system so as to trigger on the embedded-sentence case -- Cyclic node: S or NP.
Second, only the features of items in the buffer play a part in feature generalization. This is because making a variety of rule discriminations correctly depends upon labelling the active node and cyclic node above it. Having used these two nodes to encode distinctions, it is generally a step backwards to wash out the distinctions by feature intersection.
Briefly, the program generalizes rules only when positive evidence is received that initiates rule merger -- that is, when another instance of the rule is created based on a (different) grammatical example sentence. Pattern merger proceeds by set intersection of the features of each pattern, governed by hierarchic conditions imposed by the context-free base. This approach is not without flaws; see C.L. Baker, *The Projection Problem*, Linguistic Inquiry, 10, 1979. For a full discussion of the generalization issue, see Chapter Three.

Having given specific examples of the acquisition procedure in action, perhaps it is best to close this section with a brief, but complete listing of the algorithm as it is implemented. For an expanded version of the same listing, see Chapter Three.

Step 1. Read in new (grammatical) example sentence.
Step 2. Attempt to parse the sentence, using modified PARSIFAL parser.
2.1 Any phrase structure schema rules apply?
    2.1.1 YES: Apply the rule; Go to Step 2.2
    2.1.2 NO: Go to Step 2.2
2.2 Any grammar rules apply?
    (<pattern> of rule matches current parser state)
    2.2.1 YES: apply rule <action>; (continue parse)
        Go to Step 2.1.
    2.2.2 NO: no known rules apply;
        Parse finished?
            YES: (Get another sentence) Go to Step 1.
            NO: parse is stuck
        Acquisition Procedure already invoked?
            YES: (failure of current parse or
            acquisition) Go to Step 3.4 or Steps 3.2.3/3.2.4.
            NO: (Attempt acquisition) Go to Step 3.
Step 3. Acquisition Procedure
3.1 Mark Acquisition Procedure as invoked.
3.2 Attempt to construct new grammar rule
    3.2.2 Try *attach*
        Success: (Save new rule) Go to Step 3.3
        Failure: (Try next action) Go to Step 3.2.3
    3.2.3 Try to *switch* first and second buffer cell
        items.
        Success: (Save new rule) Go to Step 3.3.
        Failure: (Restore buffer and try next action)
            Re-switch buffer cells; Go to Step
            3.2.4
    3.2.4 Try *insert trace.*
        Success: (Save new rule) Go to Step 3.3.
        Failure: (End of acquisition phase) Go to Step 3.4.
3.3 (Successful acquisition)
    Store new rule: <packet, pattern, action>
    If another rule in this packet with
    same <action>, generalize by intersection of buffer features.
    Go to Step 2.1.
3.4 (Failure of acquisition)
    3.4.1 (Optional phrase structure rule)
        (Continue parse) Advance past current
        phrase structure rule; Go to Step 2.1.
    3.4.2 (Failure of parse) Stop parse; Go to Step 1.

1.3 Accomplishments and Limitations

What then are the contributions of the research? An easy way to classify the results is to group them into one of two broad concerns of artificial intelligence research:

> *Engineering.* Can we actually build a system that achieves the specified performance; in this case, acquires syntactic knowledge?
>
> *Cognitive science.* Can we obtain better explanations (build better *theories*) for human abilities, in this case, how people acquire their syntactic knowledge?

Both aims have been furthered by the results of this research. Let us consider first the engineering side, and then take up the intriguing question of human competence.

1.3.1 Engineering

RULES ACQUIRED

In some circles, the ultimate criterion for a successful engineering project is simply whether a proposed implementation *works*. How many rules can the system acquire?

Grammar Rules

A good standard for scoring the success of grammar rule acquisition might be to compare the rules that LPARSIFAL can acquire on its own against the rules that were hand-coded by Marcus for the original PARSIFAL. Perhaps the most surprising result here is that the "stupid" acquisition procedure can acquire a rich and varied set of syntactic rules, as indicated below and in Figure 1.6. Many of the "core" set of rules in Marcus' grammar can be acquired.

Highlights of the acquired rules include:

--Imperatives:	*Throw the ball!*
--Affix hopping:	*Mitch should have been kissing Sue.*
--Adverb preposing:	*Never have I seen such a mess!*
	There goes the truck!
--Passives:	*The glass was broken by Mitch.*
--Simple *Wh* movement	*Who did Mitch kiss?*

Many of these rules correspond to the acquisition of the proper grammar rule actions for base structures expanded via the X-bar schemas (for example, the *do*-support rule that checks whether *do*

can be attached as an auxiliary verb.) The fleshing out of base rule schemata turns out not to be straightforward; as we shall see, local context for each must be remembered and properly generalized. The other rules of Figure 1.6 fall into two classes: local and non-local rules. Local rules include auxiliary-inversion, affix-hopping, and adverb preposing. They are accomplished via *switch*, *attach*, and *insert lexical item*, the local operators. Among the non-local rules are passive and simple *wh*-movement, implemented via *insert-trace*. Some rules, such as the adverb preposing rule that copes with a topic modifier like *never* at the front of a sentence, are originals that were not in PARSIFAL's set of grammar rules.

Phrase Stucture Rules

Many major phrase structure rules can also be acquired by expanding the initially provided N and V schemas. These include rules for Noun Phrases, including adjectives, articles, and some noun complements; Verb Phrases, including some verb complements; substantial details of the English auxiliary verb system. Other categories, such as Prepositional Phrases, could be acquired via simple extensions of the X-bar procedure as discussed at the end of section 1.2.1.

It also appears that the acquisition of embedded sentences with explicit complementizers, e.g., *For Sue to kiss Mitch would be interesting*, is within easy grasp of the current system. The key here is an extended use of the X-bar system described in footnote 17 above.

In brief, the procedure works reasonably well within the realm of its design; many of the grammar rules written by hand in Marcus' system can be acquired automatically.

Rule	Description of situation handled
unmarked-order	simple declarative sentence, NP-VP
aux-inversion	inversion of auxiliary and subject
imperative	missing subject in imperative sentence
modal	affix hopping
future	
perfective	
progressive	
do-support	use of auxiliary *do*
passive	simple passive sentences with *by* phrases
wh-move	simple *wh* movement, *Who did Mitch kiss?*
insert-to	insertion of *to* e.g., *I helped John to do it.*
objects	objects of sentences
parse-pp	prepositional phrases
noun	noun parsing
propnoun	
propname	
parse-det	determiners
parse-adj	adjectives
s-done	final punctuation
adverb-pre	negative adverb preposing (*Never have I such a mess*)

Figure 1.6 - LPARSIFAL acquires a wide range of rules.

COMPUTATIONAL EFFICIENCY

Although questions about the efficiency of a procedure should probably not be adressed until the fundamental issue of descriptive adequacy has been settled, in some ways the acquisition procedure depends fundamentally upon constraints that (trivially) ensure efficient, as well as successful, acquisition. There are two brief points to be made here.

Use of partial parsing

By adopting Marcus' Determinism Hypothesis, as much of the sentence as possible will be parsed without invoking the acquisition component. This means that as much already-built structural context as possible is made available as a source of information for the acquisition procedure.

Finite number of rule hypotheses

The space of possible grammar rules itself is finite. Consider the entire set of new rules available for hypothesis when a bottleneck has been encountered. Any rule consists of a finite number of actions (three or so) conjoined to a *pattern*. The pattern in turn is composed of set of feature tests defined over five cells (the three buffer cells, the current active node, and the cyclic node above the current active node). If the number of feature tests is finite (i.e., the total length of a pattern is finite), then the total number of possible grammar rules ever available to the system is the cross-product of the number of rules times the number of possible patterns. What is the cardinality of the set of feature tests? Clearly, it is finite, for both practical and theoretical reasons. Practically speaking, since the total length of any grammar rule pattern is finite -- bounded by address space of the machine -- the set of all patterns is finite. This is hardly "proof" that the set of patterns is finite *in principle*, however. More to the point, the features tests deployed are all either checks for category membership, e.g., *is the current active node of type NP?* or else for specific features of lexical items, e.g., *is book singular or plural?*. Since almost all proposals for context-free phrase structure base of natural languages have assumed a finite number of phrase structure categories there are only a (small) finite number of possible category checks. Demonstrating that there are only a finite number of possible "specific lexical features" is somewhat more difficult, but indications are that it too can be independently established.[29]

29. Such proofs could be based, for example, on connections to "semantics"; e.g., a theory of "Substantive Universals"; see Pinker, Grimshaw, and Bresnan [to appear].

1.3.2 Cognitive Science

Research in artificial intelligence has been claimed as a window into human competence, a way of constructing theories about "mental computation." Here, the achievements of the research are not so clear-cut. The chief difficulty is that our theories of mental computation can be no stronger than our theories about computation in general and our methods of evaluating (and verifying) competing computationally-based theories. Existing computational theories of language use or language acquisition falter on both counts.

On the one hand, it might be argued that our existing models of computation -- at least as applied to language -- are far too specific. For example, many computational linguists employ the tools of "natural" complexity measures as a way to evaluate competing algorithms for language processing; such measures are inherently based on the step-counting performed by serial processors. Given our relative ignorance about the machinery of the brain involved in language processing, it would seem wise to develop a *machine independent* theory of computation. To evaluate processing theories of language use with measures that depend crucially on possibly false assumptions -- say, serial processing -- is to insist on the wrong evaluation measures. In this view, the confirmation of computationally-oriented theories about language acquisition (and processing) should come "from above", that is, from a more abstract notion of computation. Here one could perhaps exploit the machinery of *abstract* complexity theory as developed since Blum [1967].[30] Of course, one could take an even weaker stance, and forego *any* claims about the connections between one's characterization of "knowledge of language" and how that knowledge is actually put to use. Chomsky's reliance on *competence* theories (as opposed to theories of language use, or *performance*) exemplifies this position. A competence theory is intended to abstract away from the domain of computation altogether; it provides only *a-computational* conditions that the neural structures for language must meet. In this sense, Chomsky has made the weakest possible commitment to a particular computational device for language processing.

On the other hand, confirmation of computationally-oriented linguistic theories might come "from below" -- that is, by using data drawn from empirical observation. While such data as measurements of processing load and reaction time are in principle available to the computational linguist, for the most part the predictions offered by current computational theories of language are too broad to be

30. In fact, the obvious proposals have already been attempted. Feldman (1972) adopted the Blum complexity approach wholesale as a way to measure grammatical "simplicity"; for example, he used derivational complexity as a Blum measure, providing a kind of "Occam's razor" on grammar selection.

effectively separated by such data.[31]

The dilemma is that one would prefer falsifiable computational theories of language, set at a level of abstraction corresponding to available evidence. Unfortunately, many current computational linguistic theories somehow strike just the wrong middle ground. Still, there is at least a gross level of "psychological plausibility" that can be used as a yardstick to evaluate language acquisition theories. Perhaps the most important measure for a psychologically interesting theory is that it be at least compatible with what is known to be true of children's acquisition of language. Here, the current model gets high marks.

Foremost among these plausibility conditions -- though it would seem so obvious as to hardly need stating -- is that what is acquired should in fact correspond to an adult's "knowledge of language". That is, the model must be powerful enough to actually acquire (an ability to parse) natural languages. Put another way, the final states attained by the model should at least be plausible representations of an adult's linguistic abilities. After all, if a theory of language acquisition does not account for how language is acquired, then what else is it for? Remarkably, many proposals in the literature do not meet even this simple demand. For example, this criterion immediately rules out proposals that can attain only representations with a weak generative capacity equivalent to that of finite state machines.[32] In particular, it dismisses procedures that can acquire only the competence to correctly order two or three word strings. Such finite combinatory devices are well known to be *inherently incapable* of representing adult linguistic knowledge.

In contrast, the acquisition model developed in this research has at least the potential to acquire the "right" sort of final state, namely, a parser for the observed syntactic phenomena of English (and, given additional research into X-bar theory, perhaps other languages as well).
This would seem to be a significant step beyond models that can acquire only the rules to concatentate three or so words.[33]

---

31. There are exceptions, particularly where the choice is between theories whose predictions about the time and space resources required to perform a particular computation are vastly different -- finite vs. infinite. Assuming that the brain has finite resources, a choice can then be made. But unless the alternative theories diverge in this way, in the case of language hopes dim that enough will be known about the relevant computational-psychological details -- independent of probably wrong assumptions about the machinery of the brain -- to distinguish between them.
See Berwick, **Computational Complexity, Evaluation Measures, and Learnability** [1980 in preparation] for further discussion.

32. Recall that *weak generative capacity* refers simply to the set of strings that a device can produce; *strong generative capacity*, to the set of structural descriptions (labelled bracketings) of those strings.

33. However, it seems likely that the current model will be unable to acquire the rules to parse strictly context-sensitive constructions. This is poses a problem. It would be an even more serious problem if natural languages depended heavily upon context-sensitive machinery. Fortunately, however, they apparently do not. See Joshi and Levy [1977]; Gazdar [1979]. As often pointed out by Chomsky [1965 page 62], the question of weak generative capacity is largely orthogonal to the issue of learnability. For example, all other things being equal, the X-bar theory contributes far more to easy learnability than any gross change from context-sensitive to context-free generative capacity.

Besides being potentially powerful enough, the acquisition model of this research also meets four other "plausibility" criteria as set forth by Pinker [1979]:

★ **Appropriate Input.** Any plausible acquisition model should use just the input data that can be granted to be available to the child. Children appear to receive only positive example sentences as reinforcement for syntactic inductions. So does this model.

★ **Finite Convergence Time.** Children appear to converge to the "right" grammar for their language rather rapidly, without (in general) making gross phrase structure or major transformational mistakes. In other words, convergence is in *finite* time, rather than probabilistically in the *limit*. As the short section on computational complexity above showed, the proposed model can hypothesize only a finite number of grammar rules; it too fixes upon its grammar in a non-probabilistic manner, in finite time. Of course, convergence times for the model cannot be construed as carrying any psychological import beyond this gross level of confirmation.

★ **Developmental Fidelity.** Any psychologically interesting model should at least roughly reproduce the developmental course of human acquisition, both its staging and its errors. As to the stages in acquisition, although the data is far from clear, children's abilities unfold in a general simple-to-complex fashion, from the capability to handle two or three word sentences, to simple sentences, to more complex inversions and embeddings. Once fixing upon major components of a grammar -- such as basic constituent order as imposed by phrase structure -- they do not generally renege. That is, children appear to acquire grammars rule-by-rule, building upon what they already know. Children's errors largely center around mistakes with affixes and morphology (e.g., *I goed home*) or obvious omission of proper feature tests (e.g., for agreement between subject and verb) rather than gross transformational mistakes.

Here too, the current model fares well. It also incrementally refines its rule base, adding at most one rule for each invocation of the acquisition procedure. "Radical reorganization" of its knowledge is not possible. The model acquires rules to handle gross aspects of phrase structure -- two and three word sentences -- before an ability to parse auxiliary verb inversion or passives; given a sentence beyond its current ability, it simply parses what it can, and ignores the rest. Refinement of already known rules lies mostly in fine tuning of the features brought into the trigger patterns for rules, thus at least leaving open the possibility of the same kind of apparent fine tuning that children go through. In fact, the current procedure generally goes through three stages in acquisition of a grammar rule: (1) an overly specific stage, where the rule is tailored to trigger exactly in the appropriate circumstances; (2) an overly general stage,

where the conditions on the specific rule are relaxed; and finally (as more evidence is accumulated) (3) the reappearance of some specific conditions on the rule's execution. Although the comparison should not be taken too seriously, this sequencing of specific-general-specific is strongly reminiscent of observed (rule-governed) linguistic behavior in children.

★ **Cognitive Capacity.** Any plausible model should not require more memory, attention, and other "cognitive factors" than can reasonably be demanded of a developing child. Given our ignorance about just what the cognitive capacities of children are, it may be inappropriate to state this condition more precisely. However, the following maxim would seem to be sensible: a language acquisition model should make the weakest demands possible on memory and attention, compatible with the goal of actually acquiring a rich enough language. In this light, the current model does quite well. It assumes no memory for past sentences; only the current sentence, as well as the rule database, figure in the construction of new rules.[34]

A major goal for future research will be to evaluate more carefully the faithfulness of the current model to these criteria.

34. In contrast, the requirement that the procedure be able to store a full rule database without memory loss is a reasonable idealization for both the program and people; without it, one could probably neither approach nor retain adult linguistic abilities.

1.3.3 Limitations and the Future

The research succeeds in its basic aim -- demonstrating that a weak, but psychologically interesting procedure can successfully acquire syntactic knowledge. However, this success should not be taken as a sign that all problems have been solved. The current model is in some sense both incomplete and incorrect, failings that arise for practical and theoretical reasons.

The *theoretical* limitations are of two sorts. First, this research has concentrated largely on syntactic issues, a focus that leaves untouched most questions about the interaction between purely syntactic knowledge and other cognitive systems. This point deserves some comment.

The goal of the syntactic component as defined in this research is to provide an ability to map between a surface string and a more abstract structure from which one can recover whatever information is necessary for "semantics" -- being deliberately vague now about just what "semantics" might be. Some have proposed that the information to be encoded for semantics should include a kind of "predicate-argument" structure (in the Fregean sense), a pairing of name-like arguments to predicates. This view would hold that the minimum representation of a string such as *Sue kissed Mitch* should be an abstract structure such that the corresponding predicate form, *Kiss (Sue, Mitch)*, can be directly "read off". It also seems useful for the representation so generated to capture some notion of *thematic role*, e.g., that in *Sue kissed Mitch*, *Sue* is the *Agent* of the predicate *kiss* and *Mitch* the *Recipient*.

If this much is so, then part of the acquisition of language must involve the system of predicate-argument relations, notions of default thematic role assignments, and methods to distinguish among arguments. At the very least, for example, we must learn that some verbs do not require a Noun Phrase object: *Sue cried.* Some of the constraints that might aid in the acquisition of such knowledge could come from *compatability* restrictions with syntactic structures; we might use the X-bar theory as a way to support proposed restrictions on possible thematic roles. In the other direction, thematic role constraints might play a part in restricting syntactic possibilities, and so aid acquisition. In any case, this entire area remains open for study.[35]

Second, it is important to stress that the current system acquires only *knowledge about how to parse.* There is good reason to doubt that this provides a full characterization of a person's "knowledge of language." To see this, consider the class of "Garden Path" sentences:

The horse raced past the barn fell.

---

35. If we add the requirement that the information encoded should include a representation of quantifier scope, focus, or presupposition, then these items too become a topic for analysis via their interaction with the syntactic component.

Though this sentence is fully grammatical (compare, *the article published in the journal stank.*), if given to the current acquisition procedure the program will simply charge ahead as people do, building *the horse* as the subject Noun Phrase and *raced past the barn* as the Verb Phrase:


```
Input string: fell
```

But this is all wrong -- the program has been led down the garden path -- since it leaves the parser holding onto the verb *fell* and nothing to do with it. As the reader may verify, the acquisition procedure too will fail, and the sentence will be rejected. Since in contrast most people (eventually) can handle such sentences, the current acquisition program is incomplete.

The source of the difficulty is the procedure's insistence on no backtracking. Since the locus of the parser's construction efforts generally moves left-to-right through the input string, it gets only one chance at resolving a local predicament. Intuitively at least, people seem to handle the garden path sentences by "backing up" and re-attempting a parse. Without taking this intuition too serously, the ability to re-try a parse in a "careful mode" would seem to be a useful addition to the current acquisition procedure as well.

There are many possible ways to add in such a facility. One obvious method incorporates the theory of predicate-argument relations described above. The basic idea is that the annotated surface structure is actually given an *interpretation* to be checked for validity against the situational context at hand (or, rather, some independently assigned interpretation of the situation). For example, in the *horse raced* case, if some "thematic component" assigned the main predicate structure of the sentence as *Raced (Horse)*, but the learner (somehow) reconstructed from situational context the structure *Fall(Horse)*, then this clash of parse against predicate could serve to signal a re-try, perhaps with particular pointer to the construction of the main Verb Phrase.

Since the gist of the parser's error above was a failure to look far enough ahead and discover that there was a choice between the verbs *raced* and *fell* as main verbs for the sentence, the error most likely would have arisen from using to broad a rule trigger pattern -- i.e., too few cells or too general a set of feature tests. To remedy this, the recovery phase would probably find it useful to automatically expand the number of cells its known rule patterns should consider.

Following lines of this sort, one is in fact led to what Marcus called *diagnostic rules*. The next few paragraphs sketch out the beginnings of a theory of such rules. The sketch is just that: a beginning.

It is not intended as final word on how such rules might be acquired, but rather to suggest ways that the acquisition procedure could be extended to deal with the problems of lexical ambiguity and conflicting rules.

In Marcus' thesis, *diagnostic* rules were typically pairs of grammar rules that adjudicated between alternative category labellings for lexical items. For example, consider the sentences presented earlier in Section 1.2.2 (page 20):

```
(Question)    Have the boys who missed the exam taken the exam today?
(Imperative) Have the boys who missed the exam take the exam today!
```

The token at the beginning of each string, *have*, can be either an auxiliary verb (as in the first sentence: *the boys have taken...*) or a main verb (as in the second sentence: *(You) have the boys...*). The choice -- really a decision between alternative phrase structure categorizations for *have* -- depends upon the morphology of the verb after the Noun Phrase *the boys*: if it is tensed, as in the first sentence (*taken*), *have* must be an auxiliary verb; an untensed *take* indicates the alternative choice.

The *en* ending thus serves as a diagnostic for the categorization decision; the patterns of the grammar rules Marcus wrote to encode the diagnostic look exactly alike except for the presence of the tell-tale diagnostic aid:

```
Rule 1(Aux-verb   have): [have][NP ][verb +en]
Rule 2(Main verb  have): [have][NP ][verb tenseless]
```

The acquisition procedure will typically acquire Rule 1 above almost as given -- except its trigger pattern will be one appropriate for a generalized auxiliary inversion rule, with no conditions to be met on the third buffer cell:

```
[Auxverb, verb][NP][  ]
```

Since sentences with auxiliary inversion can safely be assumed to be encountered before rarely heard sentences where Rule 2 is applicable, let us assume that some such aux-inversion rule as listed above is known to the system. Now when a sentence like *have the boys take...* comes along, the existing and overly-general aux-inversion rule will erroneously trigger. *Have* will be attached the auxiliary verb of the sentence, a real error.

One way out of this dilemma is to invoke the back-up procedure sketched above: assume that the error is detected by a failure to match situational context against the interpretation of the annotated surface structure corresponding to the faulty parse. In the case at hand, since the learner has made the serious mistake of confusing a question with a command, there should be no trouble at all detecting that something has gone wrong. (For example, any attempt to respond felicitously to the sentence will probably go astray if its "meaning" is so severely misconstrued.) Having detected an error, the system will attempt a re-parse, but with the following difference: it uses the (faulty) parse tree just built to look for alternative categorization choices that might have led it astray. This is just the right approach, because it appears that many diagnostic rules (and "garden path" problems) arise out of faulty category assignments -- classifying an item as an auxiliary verb rather than main verb, for example. Again considering the specific example sentence troubling the acquisition procedure, we see that its (incorrect) parse tree has labelled *have* as an auxiliary verb, whereas the dictionary entry for *have* indicates that it can be a main verb.

Suppose then that the acquisition procedure, knowing it has gone astray and knowing that its first (and only) alternative lexical categorization is to make *have* a main verb, does so. Assuming that the standard acquisition procedure can take matters from there -- and discover that *you* must be inserted as an understood subject -- [36] then this choice will eventually succeed. We must further assume that, since the procedure is working in a "careful" mode, it saves the feature details of everything it can get its hands on -- that is, the features of items in all three buffer cells:

```
pattern: [have][NP][take tenseless]
action:  insert you
```

This is the right answer -- given one additional stipulation. The patterns for the rule just built and the existing aux-inversion rule conflict. Clearly we do not want to merge the two patterns -- and indeed, since the two rules have *different* actions (*switch* and *insert* ⟨*you*⟩) the acquisition procedure does not merge them. But how can the procedure decide which of the two should have priority? Note that both rules still match against the string, *Have the boys take the.....*

To settle this dispute appeal is made to the general principle that *more specific rules should trigger before more general rules.* In this case, the *insert* rule is the more specific, since it refers to all three buffer cells and the *switch* rule to just two. This gives the right result, but one might question the validity of the additional stipulation. Actually, the constraint can be motivated on quite general

36. This is not straightforward; for details, see Chapter Three.

grounds. The triggering of specific before general rules is a widely recognized principle in both the linguistic and production system literature.[37]

In addition, without such a principle it is difficult to see how a diagnostic rule could ever trigger -- unless some explicit information about rule ordering were placed into the system. But explicit ordering is exactly what is to be avoided. As Baker points out [1979], any such extrinsic ordering information may demand negative evidence for its acquisition. In a system where rules can be optionally or obligatorily applied, in order to learn that Rule A must precede Rule B an example must be presented showing the mistake of the reverse Rule B-Rule A ordering. The procedure presented in this research side-steps this difficulty by being ordering rules *intrinsically*: rules are fired in a data-driven fashion based upon the current context of the parse -- and the specificity principle -- and not upon any explicitly coded information.

It is interesting to note that the addition of the specific-before-general device also provides a way to eliminate incorrect rules. This is important, because up till now the procedure could always add new rules to its database but there was no provision for removing rules. To dispose of an unwanted rule, one simply formulates another rule (the "right" rule) with an appropriate "more specific" pattern, and the wayward rule will never be able to trigger.

So far, limitations hinging mostly on the system's inattention to lexical disambiguation have been discussed. There is a final limitation of the current acquisition procedure that falls out of a specific design decision. For the moment, there is but one grammar rule action that deals with displaced constituents: *insert trace*. Though this action was designed to deal with movements of Noun Phrases, it also handles other constituent movements, in particular, *wh* movements, as in,

Who did Mitch kiss $_{\text{trace}}$.

But this is a potential problem. Since there is no distinguished device to keep Noun Phrase and *wh* movements separate, there can be no way to distinguish the traces each leaves behind. In sentences where both sorts of movement occur, this can lead to difficulties in interpretation. Consider the

---

37. In linguistics, the principle has been explicitly formulated by Kiparsky [1973] and Lasnik and Kupin [1977]. In production systems, exactly the same proposals have been advanced by McDermott and Forgy [1978] and Rychener and Newell [1977]. Alternatively, as the Rychener and Newell article notes, one might simply impose a *lexicographic* ordering on the set of production rules, and stipulate that recently formulated rules take priority over older rules. If the more-specific diagnostic rules are always constructed in response to the failure of more general rules, then this tack amounts to roughly the specific-before-general constraint.
In fact, Braine [1971] has proposed a language acquisition model based upon the recency principle that leads to just this "specific before general" ordering for rules.

following sentence from Fodor [1978], where the Noun Phrase *you* and the *wh* word *who* are both displaced:

> **Who did you expect** $_{\text{trace}}$ **to make a potholder for** $_{\text{trace}}$
>
> A sentence presumably derived from some form such as,
>
> [*did you expect* [**you** *to make a potholder for* **who**]]

Without some additional mechanism, there seems to be no way to discover the proper bindings of the two traces; how could one know, for example, that the sentence did not mean, *did you expect who to make a potholder for you*?

In fact, there is good linguistic evidence that *wh* movement is not subject to the same restrictions as Noun Phrase movement. If so, this too would suggest positing distinguished machinery for *wh* movement. This is just what Marcus did in the original PARSIFAL; there was a separate stack to hold displaced *wh* phrases. (This proposal was in fact first made by Woods [1969] for his ATN parser.) An obvious extension to the acquisition procedure is to factor the distinguished *wh* stack back into the picture; at present, the lack of such a device means that interactions between *wh* and Noun Phrase movements cannot be acquired, nor certain cases where *wh*-movements do not comply with the restrictions on Noun Phrase movement.[38]

38. A variety of other syntactic rules have not been tackled. For example, *there*-insertion (*There seems to be a lion sighted.*) troubles LPARSIFAL. However, good evidence exists that such sentences are generated by the base phrase structure component. See Ochrle [1974]. In fact, the inability of the acquisition procedure to handle *there*-insertion might be taken as good evidence that such forms are base generated.

1.3.4 Connections to Linguistic Theory

With these accomplishments and difficulties in mind, it should be noted that the aim of the thesis is not to develop a computer program that can acquire rules to handle all syntactic phenomena. Although this is a laudable goal, its feasibility appears closely yoked to our understanding of those phenomena. For example, if there is no satisfactory descriptive account of conjunction, then explaining how to acquire conjunction seems futile. Without knowing what structure is to be acquired, the learner is at sea. Instead, the syntactic abilities that are within the acquisition program's grasp have been carefully scrutinized, to see how the processing restrictions imposed toward computational ends compare to the structural constraints on generative grammars proposed by linguists.

Perhaps most importantly then, the computational assumptions of the acquisition procedure parallel the structure of two of the most tightly formalized versions of transformational grammar, those of Culicover and Wexler [1980] and Lasnik and Kupin [1977]. The match is laid out briefly in Figure 1.7 below; for a complete discussion, see Chapter Four. On the left hand side of the diagram are some of the assumptions and restrictions of the linguistic theories; on the right, those of the acquisition procedure. Even though the linguistic work was done independently of this research, the two sets of stipulations are identical.

Culicover and Wexler assume a simple learning procedure that postulates only one new rule at a time. From this starting point, quite similar to the LPARSIFAL's, they attempt to arrive at constraints sufficient to ensure that the procedure converges to the adult's transformational grammar in a finite time. Importantly, a further assumption is that the learner's grammar must be acquired on the basis of only *simple* data, that is, grammatical examples whose depth of sentence embedding is two or less. Culicover and Wexler achieve convergence in finite time by keeping the number of hypothesized rules small, a situation guaranteed by restricting the context of rule application. Context restriction is accomplished through a host of stipulations, among them, those that Culicover and Wexler call the Binary, Freezing, and Raising principles. Without going into extensive detail about the exact formulation of these principles, their rough intent is to limit the scope of rule effects to a small radius about the rule's point of action. As Chapter Four demonstrates in detail, the constraints advanced for the LPARSIFAL acquisition procedure apparently subsume the Culicover and Wexler stipulations.

Lasnik and Kupin's [1977] mathematically formalized version of transformational grammar attempts to "present a particular theory of syntax in a precise way." As such, it stipulates certain very exact conditions on a transformational grammar, conditions compatible with many current proposals in linguistics. Their restrictions guarantee a small number of simple rule actions and uniquely applicable rule patterns, precisely the intent of LPARSIFAL. Once again, as can be seen from Figure 1.7 below,

these limitations correspond quite closely to those independently motivated by the design of the LISP program.

In brief, the following advances have been made:

★ Development of a working computer program that can acquire substantial syntactic knowledge of English under restrictions faithful to what is known about human acquisition.

★ Demonstration that *constraint* plays a crucial role in the success of the acquisition procedure. Importantly, the same constraints that ensure efficient parsing -- the Determinism Hypothesis and locality constraints on rule patterns and actions -- also play a key role in acquisition.

★ Discovery that the locality constraints proposed for the acquisition model mirror the structural constraints advanced in several current linguistic theories.

★ Formulation of a proposal for the acquisition of phrase structure rules, based upon the "X-bar" theory. Exploration into the use of the X-bar theory as a model for lexical acquisition and as a source of testable hypotheses for the actual developmental course of human acquisition.

Culicover and Wexler Constraints	Acquisition Procedure Constraints
Incremental rule acquisition	Incremental rule acquisition
Universal base (can be weakened assuming a theory of base rule base rule acquisition)	Universal base (can be weakened assuming a theory of base rule acquisition)
NO negative external evidence	NO negative external evidence
Only current sentence used to construct new rule	Only current sentence used to construct new rule
Small number of new rules available for hypothesis	Small number of new rules available for hypothesis
Rule construction based on "simple" data: depth of embedding at most two	Rule construction based on "simple" data: depth of embedding at most two
Binary principle	Determinism plus locality restrictions imposed by buffer and active node stack
Freezing principle	
Raising principle	

Figure 1. 7 (a) Constraints advanced by Culicover and Wexler vs. those of the acquisition procedure.

Lasnik and Kupin Constraints	Acquisition Procedure Constraints
Only one action per rule, affecting at most two constituents	Only one action per rule, affecting at most two constituents
Rules not marked as optional or obligatory (no *extrinsic* ordering)	Rules not marked as optional or obligatory (no *extrinsic* ordering)
Rule patterns use only one string condition; no arbitrary Boolean conditions	Rule patterns use only one string condition; no arbitrary Boolean conditions
Subjacency	Local access to active node stack
More specific rules fire before more general ones	More specific rules fire before more general ones
Small number of actions	Small number of actions

Figure 1.7 (b) Constraints advanced by Lasnik and Kupin vs. those of the acquisition procedure.

Figure 1.7 - Comparison of linguistic and computer program constraints.

1.4 The rest of the thesis

Chapter 2, Theoretical and Psycholinguistic Foundations, aims to provide a sound underpinning for the assumptions about the acquisition model presented in Chapter One. To do so, it first outlines a basic framework to evaluate theories of language acquisition. Second, it covers psycholinguistic results about what information children receive as "input" for acquisition. Third, it briefly reviews relevant formal language theory results, and several previous computational models of language acquisition.

Chapter 3, LPARSIFAL: The Acquisition Procedure, is the heart of the thesis, presenting a full-fledged description of the acquisition procedure along with several step-by-step examples of its functioning.

Theoretical Foundations

2.1 A Framework for Language Acquisition Theories

The acquisition procedure unveiled in the introduction rests upon a number of very specific assumptions about the nature of acquisition. Among all these assumptions, a linguistic one predominates: belief that (1) the acquisition of syntactic knowledge is mediated by the extensive pre-existing knowledge of the structure of language and (2) the final state of knowledge can be represented as a set of *rules*, a grammar. Starting from this cornerstone, Chapter Two will first present a framework for (any) theory of the acquisition of syntactic knowledge, and then, using that framework as a gauge, survey previous computational, psycho-linguistic, and mathematical language learnability results.

What should a theory of the acquisition of syntactic knowledge look like? Certain ingredients would seem to be inescapable. At the very least, the learner receives some input from the environment, and then constructs (or selects) some grammar from the set of possible grammars. Finally, the very idea of a construction process presupposes a procedure to compute the desired grammar. Schematically, the situation is as pictured in Figure 2.1 below. Input I simply refers to whatever external environmental factors bear on syntactic acquisition; P to any procedure adopted to find the proper grammar g; and G to the possible class of (so far) unrestricted grammars.

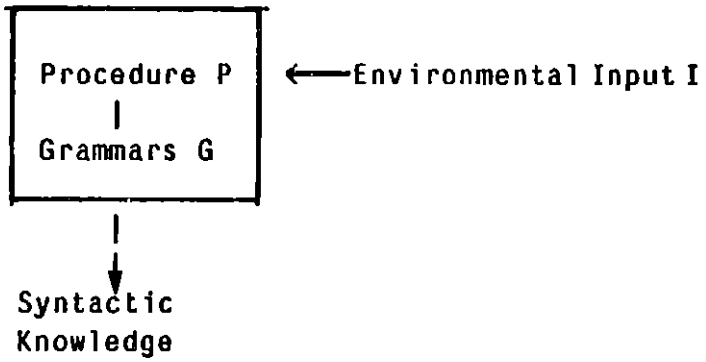


Figure 2.1 - Syntactic acquisition can be blocked into three components.

These three components -- input I, grammar G, and learning procedure P -- form the basis for a theory of language acquisition. To these three components Wexler [1978] has added three criteria of adequacy. First, the class of grammars acquired must be rich enough to cover the observed phenomena of natural languages -- that is, G must consist of a sufficient number of *descriptively adequate* grammars. Next, the external information I that the child draws upon to construct (or select) a grammar must be empirically true. In other words, input I must not specify more

information about which grammar to select than is actually available to the child. Finally, every grammar in G that is indeed a humanly possible (hence learnable) grammar can be learned by procedure P from the external information I.

The virtues of these criteria seem beyond question. Take the condition on external information: it is hard to imagine how any reasonable theory of learning syntax could require an information set I that is stronger than that available to children. Such a theory would have two flaws. First, it would no longer be an empirically true theory of human syntactic acquisition; and second, it would place the burden of selecting the right grammar on an external adult (or environment). For suppose that one rejects strict adherence to what is known empirically about the evidence children get for syntactic acquisition. Then there is nothing -- in principle -- to prevent adopting extremely powerful sorts of structured input, such that *any* grammar, no matter how bizarre, might be acquired.[1] Of course no one would in practice suggest that children learn language on the basis of such a rigid, severely structured input, where the burden of learning syntax is placed on the adults who must encode the grammar. But, as mentioned briefly in the first chapter, even a seemingly less drastic choice than a direct encoding has dramatic theoretical implications for language learning. The results of Gold [1967] show that, given only grammatical example sentences, *no* infinite cardinality language is learnable, but, by adding an *informant* who pairs each example sentence with the labels *grammatical* and *ungrammatical* almost *any* language (formally, any recursively enumerable set of recursive languages) is learnable.[2] The choice of I in any theory of syntactic acquisition thus has strong consequences for its remaining content -- G and P. If one opts for both positive and negative external evidence then the structured form of the input I suffices to ensure learnability; G and P play a minimal role. With only positive evidence, one must rely on G and P to take up the slack left open by dropping negative examples.

Now consider the assumptions behind the requirement for a descriptively adequate class of grammars G. Because one is aiming for a learnability theory of an adult linguistic ability, it seems natural that the structure acquired should correspond to what is known about that ability. Since the best theory of syntactic abilities we now have (for better or worse) is some version of transformational grammar,[3] it is clear that a reasonable theory should explain the acquisition of *grammars* -- in particular, the abilities implied by transformational grammars or some computational analogue of their capabilities. Any *other* theory of syntactic acquisition must first argue what it is, besides a grammar, that an adult acquires when such learning takes place. Although it is possible to imagine plausible options for

---

1. Consider an encoding of the grammar in the input information I, or, even simpler, an adult just *telling* the learner what the right grammar is.
2. See Appendix 2 of this chapter for a brief list of these theorems.
3. Including all its recent descendants, the so-called *Extended Standard Theory* of Chomsky, and the *functional-lexical* theory of J. Bresnan. See Bresnan [1978], [1980].

alternatives, there are none that have been worked out to the level of descriptive adequacy or even simple refutability necessary to qualify as a serious replacement for transformational grammar.

Thus, requiring that transformational grammars be acquired would seem to be simply a matter of good sense, enlisting the most powerful and structured theory we now have in the explanation of the difficult research problem of language learnability. A structured theory helps, because the more detailed and principled the theory, the more we might expect it to provide correspondingly powerful (and more easily refutable) constraints on learnability conjectures. In this fashion, the assumptions of a syntactic theory mesh to furnish very precise structural restrictions on grammars, a source of constraints and predictions that can be readily tested.

Surprisingly, this is not how work with computational models of language acquisition has proceeded. Instead of being grammatically based, several proposals have represented an adult's knowledge of syntax as part of a large network of highly interacting "packets". In these theories, the conceptual content of words is blended with proposals about causal inference and "pragmatic" information -- in short, the gamut of cognitive abilities. Although this is an initially plausible, even attractive hypothesis for language acquisition, what does this strategy imply for a theory of syntactic acquisition?

First, one is left with the job of building a complicated theory about language and human cognition, yet one that must at least duplicate the structural principles of transformational grammar. To this theory must be added a specific proposal for (at best) making precise the informal notions of "concept" and "pragmatic knowledge." But still we are not done. Because the representation of the adult state in such a theory now embraces almost all cognitive faculties, we have placed ourselves in the uneasy position of having to explain the acquisition of most human cognitive abilities. It would be as if in biology one had to explain the functioning of the heart -- and design experiments to test such explanations -- on the basis of a theory of DNA replication. In the end, because the organism is one system, it should be true that the two are at some level "connected", and the functioning of the heart "explainable" in terms of DNA.[4] But no one would imagine using this ultimate reductionist goal as a day-to-day working research strategy. One cannot imagine a more difficult scientific position.

Consequently, this research adopts Wexler's criteria of adequacy as the touchstone for theories of language acquisition, comparing alternative proposals and research by considering how each specifies the set of grammars G, the information set I, or the learning procedure, P. Existing work on language acquisition reviewed in this chapter divides neatly into which of these three, G, I, or P, are focussed

4. The notion that a reductionist strategy should be set at a level of decomposition appropriate to the explanatory task at hand has been explicated by Putnam [1973] and Fodor [1968].

upon:

--G (class of grammars): Formal language theory results

--I (information input): Psycholinguistic studies of young children

--P (learning procedure): Computational models

Although Wexler's requirements for adequacy would seem to be the bare minimum for any theory of syntactic acquisition, we shall discover in the critical analysis to follow that few other computational models have adopted even his three stipulations.[5]

2.2 P: Other Computational Models

Previous computational theories of language acquisition fall roughly into two classes: those based on general learning heuristics and those motivated from linguistic principles. The a-linguistic theories depend upon such methods as general pattern-matching procedures to induce acquired knowledge. The linguistically-oriented acquisition models in turn are of two sorts, those relying upon some specific linguistic representation, perhaps computationally-oriented, e.g., augmented transition networks [Woods, 1969] or even finite-state machines [Selfridge, 1979] for their target representation of grammatical knowledge; and those models that use statistical or distributional methods to infer their knowledge. To illustrate the general characteristics of both the linguistic and a-linguistic approaches, one exemplar of each sort is discussed below; several other examples are surveyed in Appendix 1 to this chapter.

2.2.1 Linguistically based models

Anderson's Language Acquisition System [1977] ("LAS") represents one of the most fully worked out language acquisition programs, one that does in fact attempt to attain some measure of descriptive and empirical adequacy. For input I to the program, Anderson assumes only *positive* examples paired with a "meaning" representation in the form of a semantic network of the concepts in the sentence. A lexicon is presupposed. The output of LAS is a context-free grammar, structured as an augmented transition network (ATN), that can both parse and generate a class of sentences covering the set of presented example sentences.

LAS is quite similar to LPARSIFAL in some ways. Like LPARSIFAL, LAS drives its acquisition via

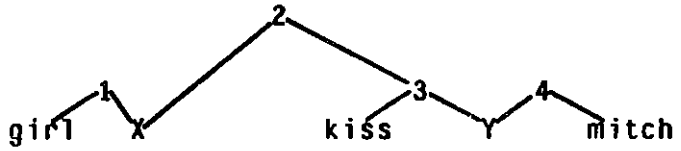
---

5. An excellent survey of computationally-oriented models of language acquisition has recently appeared. See S. Pinker, *Formal Models of Language Learning*. Cognition, 7, 1979, pp.217-283.

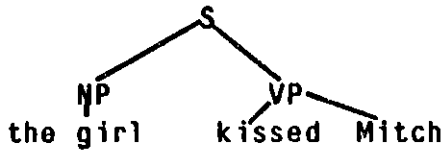
attempted analyses of positive example sentences. And just like LPARSIFAL, LAS too uses whatever rules have succeeded as the focus for its new rule-bulding efforts.

However, unlike LPARSIFAL, LAS is semantically based. Anderson assumes that the semantic representation (roughly, predicate-argument structure) of a sentence corresponds quite closely to its syntactic form. Given this partial match, if the acquisition program is assumed to have (or be able to infer from "situational context) the predicate-argument tree corresponding to the input string, it will be able to easily infer the syntactic structure of the string.

To see how this works, consider the string, *the girl kissed Mitch.* It has more-or-less the predicate-argument structure, *Kiss (the girl, Mitch),* and would be represented in the LAS semantic network roughly as,



Note that this structure -- save for the omission of items such as *the* -- is virtually isomorphic to the constituent structure for the same string:



Thus, by simply twisting the semantic network slightly, LAS can deform the semantic network it already has into the syntactic structure it wants.

Anderson must advance additional stipulations on possible semantic networks in order for LAS to proceed this far. Foremost among these is what Anderson calls the *graph deformation condition.* Briefly, this restriction is designed to ensure that semantic networks and their corresponding syntactic structures must indeed be nearly isomorphic: one cannot convert a semantic network into a syntactic one by any deformation that would necessitate crossing the links of the semantic net. In addition, since the semantic network includes no direct places for non-meaning bearing items -- articles, inflections, conjunctions, and so forth -- the program must ignore these items when it builds the semantic network for a given input string.

Given the assumption of virtual isomorphism between syntactic and semantical structures, it is easy to find assumptions in Anderson's program that reflect the syntactic constraints proposed by transformational accounts. For instance, consider Anderson's program BRACKET that does the actual job of converting an input string into its properly nested propositional bracketing. As its first step, the program actually outputs the labelled bracketing below:

```
(a) The man who robbed the bank had a bloody nose. ⇒
(b) [[the[]man[who robbed[the[]bank[]]]]had[a[bloody]nose[]]]
```

As one can see from this example, the BRACKET program presumes significant syntactic knowledge. For example, the empty pairs of parentheses in the above bracketing assumes knowledge of a phrase structure schema for Noun Phrases of the form,

```
Noun Phrase⇒(optional modifier) noun morphemes (optionalmodifier)
Modifier⇒proposition (optional modifier)
```

Given this schema, empty brackets denote unfilled optional modifier slots. Anderson mentions that this schema implies some restrictions on the structure of language, e.g., that a Noun Phrase is composed of at least a noun. Of course, this observation follows trivially from virtually every formulation of phrase structure rules; compare the X-bar theory.

Likewise, the graph deformation condition mirrors the context-free restrictions adopted in almost every careful formulation of phrase structure rules.

Once BRACKET converts the input sentence to its nested form, LAS enters its actual acquisition phase. The learning component of LAS sets up an ATN network corresponding to the bracketing provided, then labels the states of the ATN network by referring to the lexical features retrieved from the word tokens, and collapsing the new network with previous, stored forms on the basis of common category features.

At first glance, LAS's capabilities parallel the acquisition procedure of the research reported here. But crucially, the bracketing program -- the component that maps between surface string and predicate-argument structure -- is assumed.

Consider an input string where constituents are displaced from their canonical positions:

```
(a) (surface string) :   Who did John tell Bill to kiss?
(b) (canonical)      :   John did tell Bill to kiss who?
(c) (predicate-argument) (tell (John, Bill (kiss (bill, who)))
```

LAS assumes an ability to construct the canonical form (c) from the input string and situational context alone (given the graph deformation condition and certain other restrictions). The view that LAS assumes a knowledge of syntax is reinforced by its focus on just content words -- nouns and verbs -- rather than grammatical function words. It is generally agreed that function words such as *the* in *the red book* serve in the recognition of constituent phrases -- this is just how PARSIFAL works. While it is true that function words by and large do not "bear meaning",[6] they do bear syntactic weight; presumably, that is their function.

Thus, LAS really does little acquisition of syntactic transformations. Rather, LAS acquires the ability to cluster content words into equivalence classes corresponding to the *labels* on an already-constructed bracketing, and can also extend simple phrase structure rules into more complex (and recursive) expansions. In brief, LAS can acquire part of the *base component* of a grammar, not its transformational component; to this extent it overlaps with the X-bar theory of phrase structure acquisition presented in Chapter One. However, additional theoretical machinery must be imported to account for the acquisition of the rules that are quintessentially human -- rules such as Subject-Auxiliary verb inversion in English.[7]

6. Although even this is not quite so: consider the relationship between quantification and articles: *the book* vs. *a book*.
7. For a more detailed analysis of the strengths and weaknesses of LAS, see Pinker [1979] cf. footnote 4 above.

2.2.2 Induction theories

Where Anderson's approach is based on a specific computational representation of language, other strategies for language acquisition by computer presume that learning can proceed using a very general inductive apparatus, a specialized extension of string pattern-matching. In Wexler's framework, this strategy throws the burden of learning onto the acquisition procedure P and possibly the properties of the information I. Structural restrictions on the form of grammars G play next to no role -- in fact, as one might expect from a strategy grounded in induction, the game is played backwards. *All* the properties of grammars are deduced from P, rather than first determining the structure of grammars and then formulating a P. String induction thus ignores any special characteristics of human languages that might restrict the hypotheses that P must ponder; to take up this slack, theories of this type are ineluctably drawn to elaborate learning procedures or heroic information training sequences.

All "inductive inference" methods rest on a foundation of *generalize-generate-and-test* (g-g-t):

1. Start with a set of grammatical examples provided by a teacher.

2. Generalize (usually, as abstract as possible) a set of rules from (1).

3. (Optional) Prune the search for possible plausible rules in (2), using some metric (e.g., generality).

4. Generate a set of candidate strings from the rules of (2).

5. Test the candidates, submitting them for review to the teacher.

6. Iterate, returning to step (2).

Not surprisingly, this six-step procedure parallels -- with the addition of feedback -- artificial intelligence search techniques e.g.,in chess; (2) and (3) are like a *plausible move generator*, and (4) and (5) together act like an *evaluation measure* for the possibilities enumerated by (2). Just as in computer chess, the practical success of this method hinges upon good heuristics for discarding the enormous number of blind alleys opened up by the move generator. Perhaps more surprisingly, the g-g-t technique has a tradition of being periodically "discovered", even though it is quite "ancient" (Solomonoff [1958] or Miller and Chomsky [1963]). For example, Knobe and Knobe's *A Method for Inferring Context-free Grammars* [1976] follows essentially the Solomonoff technique point-for-point.[8]

---

8. See Appendix 1 for details of the Knobe and Knobe work.

Clearly -- again just as in computer chess -- the pruning heuristics of step 3 need not correspond in any way to the procedures people actually use to acquire language. In fact, ignoring the known structural constraints on human languages practically precludes psychological validity. Without such constraints, "g-g-t" syntactic acquisition algorithms must remain variations on a basic six-step theme; their incorporation of psycho-linguistically implausible training sequences and negative reinforcement seriously undermines their linguistic soundness.

Finally, what sorts of languages do these methods actually acquire? For those string induction programs that do not represent their knowledge as a grammar, evaluation poses difficulties; only the set of successfully processed strings can serve as a gauge, and such a list is usually not available in published reports. For g-g-t programs that build a grammar-like corpus of rules, while interesting context-free programming languages and *very* restricted subsets of natural languages have been tackled, no program has approached the acquisition of a *descriptively adequate* grammar for natural language.

Solomonoff: Discovering phrase structure rules [1958, 1959]

Solomonoff's idea for inducing a set of phrase structure rules from a subset of example strings presents the prototypical g-g-t strategy; several others are presented in Appendix 1. The core idea can be presented with an extended quote from his original 1959 paper:

> The method used for phrase structure languages consists of "factoring" the set of acceptable sentences into the (Boolean) union of "products" of certain sets of phrases. Here, we use the term "product" to denote concatenation. If a_i is a member of the set of phrases A, and b_j is a member of the set of phrases B, than a_ib_j (the concatenation of a_i and b_j) will be a member of the set of phrases designated by A x B. For example, suppose the set of acceptable sentences as a b, a c, b b, and b c. We could completely factor this set into the "product" of the sets A=(a, b) and B=(b, c).
> The method of factoring that is used here involves a "teacher". If it is suspected that all of the members of A x B are acceptable sentences, and only a few of these members have been given to the machine -- (say a c and b b) -- then to verify this "suspicion" the machine would have to ask the teacher if a b and b c were acceptable sentences.... Usually, it will not be possible to find a single pair of factors that yield the entire set of acceptable sentences, so we will then use the union of several such products. ... We shall assign more utility to those factor pairs that produce the largest numbers of acceptable sentences.... It is clear that we will soon have an enormous number of factor sets. We will use the utility concept to reduce this number to manageable proportions by giving prior consideration to sets of high utility. (1959, pages 7 and 8).

Note that the Solomonoff method incorporates each of the six steps of the g-g-t procedure.

Solomonoff offers as well several alternatives for pruning factor trees and refining plausible move generation. Alternative factors can be cut by a Bayesian probabilistic utility formula or by a simple numeric weighting (if |A|= number of elements in a set of phrases A then just use |A||B|). The number of factors to consider can be trimmed by adopting a suggestion of Miller's and Chomsky's [1963] to search for "cycles", ordered pairs of phrases that can be arbitrarily inserted or deleted in a string because one remains in the same "state" of the phrase structure rule. The learning program takes this invariance as evidence for phrase structure constituents, but of course it must generate each possibility and submit it to the teacher for confirmation. (The need to test a hypothesized rule via teacher confirmation is common to most g-g-t procedures; see Appendix 1 to this chapter for additional examples.)

A second, related device works in conjunction with the rule generator to hunt differentially for recursive rules. For example, suppose the presented example string was *John is the boy that kissed the girl*, and further suppose that the system could classify this string as the containing the pattern, Noun Verb Noun *that* Verb Noun. The procedure would then start deleting items from the sample string and submitting the truncations to the teacher for verification until it found the smallest unit still grammatical -- here, the sequence Noun-Verb-Noun (*John is the boy*). Next it would use this minimal grammatical set of items to generate new test combinations, testing for recursion by adding repetitions of the items previously deleted and again handing the results to the teacher for judgment. In our example, it would add back repetitions of the sequence *that* Verb Noun to its core triplet Noun-Verb-Noun; that is, it would generate strings such as *John is the boy that kissed the girl that kissed me.* If the teacher ruled the new string grammatical, the system would infer the rule, X⇒X*that* N V; X⇒N V -- about the right rule for right-branching sentences of this sort. Infinite recursion would be deduced on the basis of just a few appropriate examples.[9]

While this approach seems very powerful, as Solomonoff himself notes it relies on a deliberately designed training sequence -- including negative examples -- for its success. This is because in order to avoid missing any possibly recursive rules, the program is designed to over-generalize wildly. It does not pay attention to the *structural* characteristics of the strings so generated, but only the recursive properties of the strings as simple sequences of tokens. This spells trouble for the Solomonoff procedure because, as Chomsky was the first to demonstrate, two strings can be almost identical in terms of surface word order, and yet be radically different in meaning.

---

9. Although Solomonoff did not computer implement these procedures, a close kin of the Solomonoff method was utilized by Pivar and Finkelstein [1964] for the induction of integer string sequences. For other examples, see Biermann and Feldman, 1972; Fu and Booth, 1975.

Consider the examples from Chomsky [1965, page 22]:

```
I expected John to be examined by a specialist.
I persuaded John to be examined by a specialist.
```

Although these two strings are almost identical, and would probably be analyzed as such by Solomonoff's procedure, they clearly are different in meaning; in the first example, *a specialist* is the direct object of the verb *expected* and *John* is the direct object of *examine* (something roughly like, *I expected a specialist -- a specialist will examine John*). But in the second sentence, *John* is the direct object of the main verb (*I persuaded John -- a specialist will examine John*). The problem is one that has been noted previously: in the first sentence, the proper argument of the verb *expected(specialist)* has been moved from its canonical position. Since the Solomonoff procedure can only accept the surface order of tokens as the evidence on which to base its rules, it can draw no conclusions about displaced elements. Thus, by design, it cannot deal with an important class of syntactic phenomena; at best, it could only acquire the base phrase structure rules of a grammar.

The moral here should be apparent. A program that cannot represent important knowledge of grammatical structure cannot acquire such knowledge.

2.2.3 Systems Based on Distributional Evidence

Some of the earliest computational models for language acquisition employed a style quite different from the "generate, generalize, and test" methods. Most of these were statistically or distributionally based. That is, they drew inferences about how words could be arranged in strings on the basis of statistical or distributional evidence provided by example strings.

Typical of this line of attack was an early program by Kelley [1967]. Kelley's procedure was designed to deal with the very earliest stages of human acquisition -- the production of just two or three word utterances. The program developed a classification strategy of words by attempting to categorize the observed words of sentences provided by the outside world into two categories -- things (nouns) and actions (verbs) -- and receiving explicit positive and negative feedback about its success from the programmer. Grammatical function words were ignored.

For instance, if sample strings such as *Sue kissed Mitch, Mitch kissed the dog, the dog kissed Mitch,* and so forth were observed, a distributionally-based system would conclude that *Sue, Mitch,* and *the dog* were all in one equivalence class, since they all preceded or all (almost) followed the word *kissed.* If we dubbed this equivalence class a "NounPhrase" the such a procedure might be said to acquire a

rudimentary rule, Sentence⇒NounPhrase Verb.

Because distributional procedures such as Kelley's are fundamentally based on experience with large amounts of data without any underpinning in grammatical structure, the objections to Solomonoff's procedure apply to them as well. Perhaps more tellingly, the fundamental assumption on which distributional procedures are based presume that the important generalizations about the structure of a language can be defined on the basis of concatenations of equivalence classes of words. Although some facts about language can be captured by a concatenation analysis, its applicability as a complete model for syntactic acquisition is in doubt. Since Chomsky's early work it is well-known that rules about what words precede and follow one another cannot be a sufficient account of our "knowledge of language".

To summarize, most computationally-based theories of language acquisition do not meet the three Wexler criteria for an adequate theory of acquisition. They fail to account for the acquisition of human syntactic abilities as represented by grammars, and so do not meet the test of descriptive adequacy. The reason is revealing. Most computationally-based theories rely heavily on computational rather than grammatical constraints for their success. By systematically ignoring what is known about (human) syntactic structure, one almost inevitably arrives at a theory that is at least syntactically descriptively inadequate.

2.3 J: The Linguistic Input to the Child

Every growing child receives a barrage of experiences from the environment, and yet somehow emerges from this welter of information with the sophisticated and coherent behavior called language.

Considering this experience, one should distinguish between two sorts of information that might aid the language learner in the acquisition of syntactic knowledge. The environment might provide direct syntactic information, either in the form of grammatical sentences uttered by adults, or (possibly) negative reinforcement via correction of the child's syntactic mistakes. But, as is well known, a sentence has additional structure beyond its purely syntactic form-- namely what the sentence *means*. If the child can enlist the semantic interpretation of a sentence to determine whether the syntax makes sense, then this knowledge could furnish beneficial feedback.

Incorporating the aid of this interpretation process makes perfect sense within the framework of current linguistics. In these theories, semantic interpretation takes place in a component that utilizes one of several alternative syntactic forms, e.g., the deep structure of a sentence [Katz & Fodor, 1964], or an annotated surface structure [Chomsky, 1973]. Whether these theories are correct is of no concern for the moment. What is crucial to note is that they describe the process of interpretation as a mapping between a syntactic form and its "meaning." Therefore, if child can somehow "make sense" of an utterance even while (perhaps) not comprehending its syntax, one piece out of three (syntactic form, mapping, meaning) is now known; in principle, this information is of value in constructing the other two, that is, in learning syntax.

The introduction of a semantic interpretation process is important for another reason. Some may have felt uncomfortable at the first chapter's near silence regarding other sorts of learning that children clearly engage in-- e.g, the acquisition of what might be loosely called "concepts"-- for example, what "over" or "a red block" means. Don't these abilities play some part in learning syntax? Of course; notice that to solve the task of "making sense" of an utterance, one can invoke the entire arsenal of a child's cognitive devices. In fact, it is precisely here, in the interpretation of situations, that the enormous "conceptual" abilities of the child probably play their role. But neither is it true that these "other" abilities suffice for language, as Culicover and Wexler stress:

> ... we do not intend anything like the suggestion that syntax isn't necessary because people can understand from situations. It is obvious that adults understand the structure of sentences so that they can correctly interpret utterances even when, for example, the referents are distant in time and space and there is nothing in the non-linguistic environment to hint at the interpretation to be given to the utterance. It is the learning of this ability that has to be explained. (1980, page 2-67)

As pointed out in Chapter One, it is not the primary business of this report to engage the questions of general conceptual learning directly. Explaining syntactic acquisition seems difficult enough; whatever general abilities are necessary to discover the "meanings" of utterances are simply assumed. What then of the purely *syntactic* evidence a child employs to construct a grammar? Here one would expect psycho-linguistics-- the observation of young children-- to provide the answers. Unfortunately the history of early child language studies is an ambiguous and stormy one. Debate rages over the functional role of a mother's speech to her child. Are children "corrected" for making mistakes in language? Does a mother's simplified spoken "baby talk" (alias *motherese*) lead the child through a sort of guided learning syntax course? Settling these questions would have important consequences for any empirically motivated theory of syntactic acquisition.

We shall call the *correction* question the problem of *negative information*, i.e., of negative reinforcement (by the mother) of ungrammatical utterances by the child. (The fact that children receive *positive* or grammatical examples has not seriously been questioned.) The second issue, the effect *motherese*, presumably derives from the first-hand casual observation of mothers talking to their babies or perhaps the suspicion that child language acquisition should parallel the adult experience in learning a new language via a sequence of graduated exercises. To anticipate the conclusions, *to the best of our current knowledge, children receive little in the way of explicit correction for syntax, and small benefit from simplified motherese.*

Negative Information

Clearly parents do not present sentences to their children in a systematic or tutorial fashion, explicitly pairing examples with the labels "grammatical" or "ungrammatical." Even so, many computational models of language acquisition have ignored this fact, and stipulated such training sequences. The program reported on in this research abides strictly by the empirical evidence, and does not admit of this kind of teaching.[10] However, negative information might be presented more indirectly, in the form of explicit correction of the child's speech. Recall that this is an important point because of the enormous gap between the class of (otherwise unconstrained) grammars learnable by only positive data and the class learnable by both positive and negative reinforcement [Gold, 1967]. With only positive data, almost no (otherwise unconstrained) grammar is learnable; but negative and positive examples permit almost any grammar to be learnable. What then are the facts concerning parental correction of child speech?

Very few child language studies have confronted the issue of parental reinforcement directly; a

---

10. Negative information was also used in Winston's concept-learning program. The program built up a network-structured concept of, say, an *arch* by making powerful use of a teacher's presentation of non-arches coupled with the warning, *This is not an arch.*

summary of most of these is displayed in Figure 2.2 below. Their conclusion is nearly unequivocal: there is but *meagre* adult correction of children's syntax. Approval and disapproval are not primarily linked to the grammatical form of an utterance, but rather to its *meaningfulness* [Braine, 1971, pages 159-161, anecdotal evidence; Brown and Hanlon, 1970].

The Brown and Hanlon study in fact tested for the possibility of correction directly, comparing the proportion of syntactically right/wrong utterances (drawn from a corpus of child speech) against the corresponding occurrence of parental approval/disapproval. Statistically, there was no effect of correction. Further, mothers seemed to understand ungrammatical utterances perfectly well, and so were not likely to even respond differentially to them. On the other hand, meaningless sentences tended to evoke a correction. They present several examples that summarize the statistical results:

child utterance	parental response
Draw a boot paper.	That's right. Draw a boot on paper. (approval)
There's the animal farmhouse.	No, that's a lighthouse. (disapproval)

The remaining studies listed in Figure 2.2 concur; adults correct the *semantic*, not the syntactic errors of children. Though the final answer is not yet in, given the nearly total absence of evidence for negative reinforcement of children's syntactic errors, it would seem wise to avoid such an assumption.

	Properties of the mother	Responsiveness of the child
Correction	Brown and Hanlon, 1970	McNeill, 1966 (anecdotal)
Well-formedness and meaningfulness	Newport, 1976; Newport, Gleitman, 1978	Shipley, Smith, and Gleitman, 1969; Huttenlocher, 1975; Sachs and Truswell, 1978; Gleitman, Gleitman, Shipley, 1972
Explictness or literalness	Schneiderman, Shatz, Gleitman, 1979	Schneiderman, Shatz, and Gleitman, 1979; Shatz, 197x
Repetitiveness	Newport, Gleitman, Gleitman, 1977	Newport and H. Gleitman, 1978; Francese, Newport, and Gleitman
Expansion	Newport, Gleitman, and Gleitman, 1977	Cazden, 1979; Nelson and Bonvillian, 1979

Figure 2.2 - Studies examining possible adult influence on language growth. (from Gleitman, 1979)

Motherese: specialized input?

An oft-promoted idea is that "babytalk" -- mothers speaking in special, simplified ways to their children -- might be a major determinant of syntactic acquisition.[11] In its strongest form this proposal asks, can the external environment be so structured as to eliminate the need for constraints on possible grammars or a sophisticated acquisition procedure? A weaker result would be to discover that babytalk merely bolsters syntactic acquisition while still necessitating constraints on grammars and the acquisition procedure. This facilitation might be revealed, say, by changes in the rate of acquisition. In principle, a spectrum of surface effects might be observable, ranging from large shifts to modest ripples in the course of acquisition.

When one considers the evidence carefully, the effects of motherese appear slight. On the theoretical side, by simplifying the input to the child we only restrict the information available to construct a full grammar:

11. The basic framework for this section is adopted from Culicover and Wexler's discussion, Chapter 2 [1980].

> Simply, less information is being given to the learner than before. . . . information is being restricted. Thus, limiting input will make a stronger nativist case, rather than a weaker one. . . .
>
> We do not mean to claim that sequential characteristics of the input can play no role in learning. Rather we are claiming that such aspects of the input cannot play such a major role in learning that there is no need for special linguistic constraints.
>
> [Culicover and Wexler, Chapter 2]

Of course, one might object that perhaps a (supposed) carefully tailored sequential form of motherese could ease the path of syntactic acquisition, leading the language learner by short steps to adult grammar. However, here too the evidence points the other way. The most thoughtful empirical work demonstrates that motherese does not greatly aid the course of syntactic acquisition -- despite claims made to the contrary. Here, as Culicover and Wexler state, there are three possible empirical findings which

> would demonstrate the crucial role of [motherese] in language acquisition, doing away with the need for special structural principles: (1) speech to children is simple (compared to speech to adults) (2) speech to children becomes more complex as a child's psycholinguistic abilities increase (in a causal sense) and (3) the more that a mother uses the special (simple) properties of [motherese] the more will her child develop language.

The best, most careful consideration of all three questions is to be found in Newport, Gleitman, and Gleitman, *Mother, I'd rather do it myself: some effects and non-effects of maternal speech style*, [1977]. In particular, they examined closely a whole host of complex statistical issues that arise in developmental studies of this kind, most prominently the possibility of spurious correlation. This error comes from mistaking a correlation for a causal relationship, here the possibly incidental connection between characteristics of maternal speech and child language growth. For example, a child's language abilities presumably grow faster when a child is younger, and at the same time the mother's speech becomes more "complex" (at least in the sense of length). The two factors are thus highly correlated, but there need be no *causal* relationship between the two -- they simply co-vary. This fallacy is distinctly possible with motherese effects, *unless* other intervening developmental variables are held constant. Most studies have neglected this pitfall; Newport, Gleitman, and Gleitman focussed upon it:

> Notice that the finding that Motherese exists cannot by itself show that it influences language growth, or even that this special style is necessary to acquisition-- despite frequent interpretations to this effect that have appeared in the literature. (page 112)

When Newport, Gleitman, and Gleitman controlled for other possible variables influencing language growth, they discovered that motherese shaped the child's language acquisition only modestly:

> ...certain highly limited aspects of the mother's speech do have an effect on correspondingly limited aspects of the child's learning. Many other identifiable special properties of Motherese have no discernible effect on the child's language growth. The maternal environment seems to exert its influence on the child only with respect to language-specific structures (surface morphology and syntactic elements that vary over the languages of the world), and even then only through the filter of the child's selective attention to portions of the speech stream. . .
> (page 131)

The assumption that the simplicity of a mother's speech to her child is asset for learning language is likewise just an assumption. After trying out a variety of different measures of "simplicity", they conclude that motherese contains *more* optional movement and deletion transformations than adult speech:

> Overall, then, "syntactic simplicity" is a pretty messy way to characterize Motherese.
> (page 151)

Newport, Gleitman, and Gleitman therefore reported,

> . . . The point is that demonstrating that speech to children is different from other speech does not show that it is better for the language learner. Most investigators have jumped from the finding of a difference, here replicated, to the conclusion that Motherese is somehow simple for inducing the grammar. . . (page 159)

Other evidence supports NGG'S finding. Perhaps most importantly, early syntactic acquisition seems unperturbed by major disruption of the input I. Gleitman [1979] has assembled a summary of research on subjects whose input I is sharply limited compared to normal children, reproduced as Figure 2.3 below. For example, deaf children of normal parents presumably cannot receive even the example sentences that normal children can. Yet they seem to spontaneously produce via home-grown sign language ("home sign") the same rudiments of early syntactic rules as normal children, in the same developmental sequence. The course of language acquisition in other handicapped children and in other cultures (represented by the other entries of the Figure 2.3) confirms this finding over a range of input limitations.

When all of the best current evidence is thus taken together, it points to a single, easily stated result: a minimal role for input I in syntactic acquisition.

Presumed Condition	Modality: Speech	Modality: Sign Language
base/surface string pairs under normal conditions	English: Newport 1976 Tamil: Williamson, 1978* Newport, Gleitman, and Gleitman, 1977	Deaf children of signing parents: Newport et al. 1977
Diminished opportunity for determining surface structure	Newport, Gleitman, Gleitman, 1977 (certain mothers) Hard-of-hearing: Dutton and Gleitman, 1979	Isolated deaf children: Feldman, Goldin-Meadow, Gleitman, 1978
Diminished opportunity for determining appropriate base	blind: Landau and Gleitman	blind child of signing parents (no examples)

*Also Snow [1972]; Cross [1977]; Blount [1975]; Nelson [1978].

Figure 2.3 - Populations with varying opportunities to receive input. (from Gleitman, 1979)

2.4 G: Formal Language Theory Results

Because of the well-known isomorphism between formalized mathematical models of certain grammars and classes of computable functions (e.g., context free grammars and push-down stack automata), language learnability theorems can be obtained by exploiting the decidcability results of recursive function theory. Care must be taken, however. The power of the mathematics can blind one to the abstract character of the learning situations so modeled. Until the work of Hamburger, Culicover and Wexler, [1975, 1980] the formalized models deployed for learnability results did not incorporate many of the restrictions usually imposed in precise accounts of transformational grammar; as a result, most sufficiently rich grammars were found to be "unlearnable." Even so, these idealizations provide upper and lower bounds on what is or is not learnable, delimiting rough boundaries that can serve as guidelines to more empirically based theories. For example, the theorems of Gold [1967] were invoked earlier to show the power of negative reinforcement in learning; likewise, the Gold results demonstrate that, even considering an ideal learner with no memory or attentional limits, transformational grammars are not learnable from only positive data.

The earliest theorems on formal learnability derive from this seminal work of Gold [1967]; these are listed in Appendix Two below. What do these results say? If the class of languages available for hypothesis by the learner includes at least one language of infinite cardinality (that is, at least one language that consists of a potentially infinite set of strings, such as English), then the learning procedure cannot settle upon the correct language if it has only positive evidence to draw upon. The result holds even if the acquisition procedure has infinite time and computational resources at its command. The "target" set is simply to large -- even if that target is any of the usual languages of the generative hierarchy (finite state languages, context-free languages, or beyond). In contrast, if negative data is admitted, then any recursively enumerable set of recursive languages is learnable in the limit.

While numerous extensions and modifications have been made to Gold's basic results,[12] their basic import still seems clear: Brute computation alone does not solve the problem of language acquisition.

What then is to be done to achieve learnability? Turning to our tri-partite theoretical framework, the only alternatives are to modify either G, I, or P.

---

12. For a review, see Pinker [1979]. Among these are: allowing an approximation to the right language [Feldman, 1972; Biermann and Feldman, 1972]; [Wharton, 1974]; ordering the presentation of examples [Feldman, 1972]; use of filtering or evaluation measures [Wharton, 1977] in the enumeration itself. Some of these tacks lead to modest improvements: certain simple languages can be identified "in the limit." But the efficiency of these procedures is still extraordinarily low, and they still presume that the learner can store all the sentences (strings) ever encountered.

Increasing the power of P is probably futile, for two reason that Wexler points out. First, by the usual definition of effective computability (Church's thesis), we know that there is no stronger sense of *computable procedure* than the one we have now. Therefore, once we have shown that there is no effective procedure for selecting a grammar using positive-only data, all hope must be abandoned for a more powerful P that might work. Yet things are even worse than this. Not only is there no *computable* function to learn from unconstrained classes of grammars, there is simply no function at all. [Culicover and Wexler, 1980, page 2-34] Thus, moving to more powerful P's -- even non-computable P's -- is doubly fruitless.

The learning procedure adopted in the Gold proofs is unreasonable in yet another way. In the Gold paradigm, the learner hypothesizes and rejects entire grammars for a language at each step. This seems manifestly to conflict with what is known about children; they seem to more gradually march to a correct grammar for a language [Brown, 1973]. Further, the Gold method of guessing a grammar based on the entire sample of sentences requires that the learner keep in memory all the sentences (strings) encountered so far, a memory requirement that borders on the absurd.

What about I? This chapter's section on the linguistic input to the child demonstrates that, on empirical grounds, the data input cannot be enriched to include negative reinforcement (presentation of ungrammatical example sentences). However, the outlook brightens if other kinds of input enrichment are presumed. Namely, if one supposes that the child can build some structure corresponding to the "underlying meaning" of an utterance (from, say, situational context) then Hamburger and Wexler [1975] have proved that a transformational grammar *is* learnable. This amounts to proposing that the learner is presented with the pair <b, s> where *s* is the surface string of words of the sentence, and *b* the "deep structure" [or perhaps a closely related predicate-argument structure].

At last we have a positive learnability result, one at least compatible with the kinds of empirical observations that were made in this chapter's section on linguistic input to the child. Unfortunately, the Hamburger and Wexler result clashes with empirical reasonableness in another way: the example sentences required for learning are enormously complex. The number of nested clauses in the presented example sentences (their "degree of embedding") must be huge, on the order of several hundred thousand, for the procedure to work. One can safely assume that no child ever hears sentences of this complex. Even this positive result fails to pass on empirical grounds.

There is but one remaining alternative: to supply additional restrictions on the class of possible grammars that the acquisition procedure can hypothesize. As Chomsky puts it, perhaps "the child approaches the data with the presumption that they are drawn from a language of an antecedently well-defined type." [1965, page 27] The goal of this research is to fill in the details of Chomsky's statement, to find out just what antecedently defined type the child presumes a language to be.

Ideally, this means we would like to completely characterize the class of possible natural languages, determining just what properties they possess that make them learnable. Further, the orientation of this research demands that the resulting model should be computationally and psychologically plausible.

Having laid down these theoretical criteria, the next chapter outlines the design of an algorithm that is intended to meet them.

Appendix 1: Other computational language acquisition models.

There is not space enough in this chapter to mention all the models of language acquisition that have appeared in the literature. For an excellent survey of formal models of language acquisition, the reader is referred to Pinker's article in Cognition [1979].

To give some idea of the variety of other work that has appeared in this vein, below are sketched other g-g-t, distributional, and "heuristic" acquisition models.

1. Uhr, 1964

Uhr's language acquisition program is statistically-based. As summarized by Siklossy [1972],

> By a process of string matching and statistical learning, Uhr's programs attempt to translate strings from one [natural language](NL1) into strings of another (NL2). The programs are insufficiently documented to explain their structure in detail, but from the output exhibited, several limits appear: the idiosyncrasies of NL1 create difficulties for the program, and cyclical behavior instead of continuous learning sometimes develops. [page 289][13]

2. Knobe and Knobe, 1976

Knobe and Knobe's *Method for Inferring Context-free Grammars* presents a g-g-t procedure in more modern dress, viewing the problem from the vantage point of computer science, not linguistics:

> We can view the [grammatical inference] problem as an example of the general inference problem. We chose to take as broad a view as we could. Thus, the part of our work that is specific to grammatical inference rather than inference as a whole, is our notation, BNF[Backus-Naur form]. This notation has been used in many other contexts with considerable success, and we therefore do not consider it a significant restriction. ... We try to make as few assumptions as possible based on observations of the use of languages. (1976, pages 129, 130).

Given this viewpoint, it might be expected that Knobe and Knobe would adhere to a strict g-g-t approach. They do; their program:

---

13. The original reference is not reviewed here, but Uhr's program probably resembles his other work: a statistical weighting scheme to adjust coefficients.

--starts with examples supplied by a teacher

--abstracts from the examples to find plausible rules

--prunes the rules by applying a metric of relative generality:
(a) prefer short rules to long rules
(b) prefer a large terminal to non-terminal ratio
(c) prefer recursive to non-recursive rules

--tests the rules by submitting a randomly selected subset or an exhaustively generated sub-sample of strings to a teacher for review.

Knobe and Knobe introduce additional heuristics to repair over- and under-generalizations, by *backpatching*. Each time a new rule is added, the program examines all previous rules to determine whether revisions are necessary; this entails a thorough check of all rules that interact with the newly acquired one.

Through these enhancements to a basic g-g-t, the program succeeds in mastering interesting context free grammars, e.g., arithmetic expressions with simple variables and function calls. On the other hand, left untackled are complex context free grammars or the combinatorial dilemmas involved in backtracking to fix rules. As with Solomonoff's method, the program relies on a teacher, a good teacher, for its accomplishments.

3. Siklossy, 1972

ZBIE, Siklossy's program, receives as input paired natural language strings and their representation in a *functional language*, FL. FL serves in the role of Anderson's network memory representation, providing an "underlying form" that attempts to capture a kind of pictorial semantics. For example, the sentence,

```
the hat and the book are in the drawer
```

would have as its FL form,

```
(be(in((and hat book))((drawer)))
```

As its "grammar" the program outputs a set of *translation rules* that map the presented language strings to forms in FL. Siklossy's learning procedure P thus must construct new translation rules by recursively matching FL forms to NL strings. This matching occurs even down to the level of

individual words; for example, the program infers lexical entries by simply pairing off tokens in the input string with FL. Given the initial rule:

```
girl is here⟶
(be (girl) here)
```

The program will acquire:

```
boy is here
(be (boy) here)⟶
is⟷be
here⟷here
boy⟷ boy
```

Unfortunately, the program requires a 1-1 mapping between tokens in FL and NL -- it will fail on the string, *a boy is here.*

ZBIE basically acquires labels for words. This must be so, because FL forms come already parsed; the acquisition of syntactic rules is not addressed at all. Even as a token-to-token mapper, the program's repertoire is limited, mastering only simple phrases. As Siklossy states, ZBIE was simply not designed to simulate the language learning behavior of human beings.

Appendix 2: A Brief list of Gold's formal language learnability results

Theorem [after Gold, 1967]: No recursively enumerable list of recursive languages is learnable from positive-only examples.

Theorem [Gold, 1967]: Any recursively enumerable list of recursive languages is learnable from positive and negative examples (so-called informant presentation).

Theorem [Peters and Richie, 1973]: Unconstrained transformational grammars on a single phrase structure base form a recursively enumerable set of recursive languages.

Conclusion: Unconstrained transformational grammars are not learnable from positive-only (grammatical) examples (so-called text presentation).

Semi-formalized versions of the terms "learnable" "text presentation", "informant presentation", and "learning procedure":

Learnable: A procedure P learns a language L if and only if after time t_n P guesses a grammar G_i that generates L and then never changes its guess. [or changes only finitely often] (="identifiable in the limit" in Gold's terminology)

Text presentation: First, discretize time to $t_0, t_1, \ldots$ At each t_i, P is presented with only grammatical examples (postive instances of strings from the language L).

Informant presentation: At each time step t_i, P is presented with an example string from the language along with an indication that the example is or is not a member of the language L.

Learning procedure P: After each example is presented, P has the option to select a new grammar as the hypothesized generator of L or else retain the grammar currently hypothesized.

The Acquisition Procedure

3.1 Introduction and Overview to the Chapter

The first two chapters set out the particular methodology of this research:

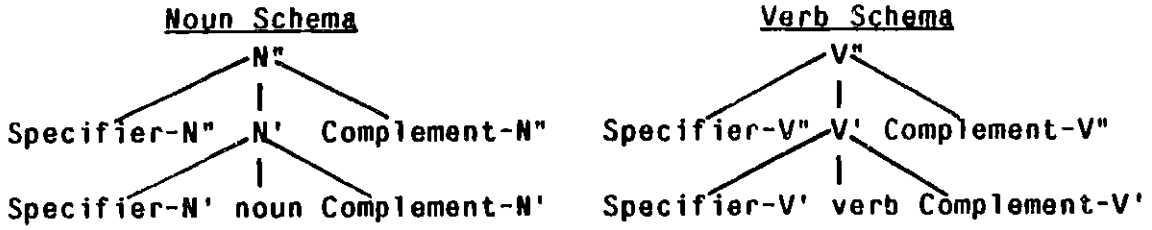
> First, to adopt the generative grammarian's view that syntactic knowledge can be represented as a *grammar* or a roughly equivalent computational form;
>
> Second, to posit that the acquisition of syntactic knowledge means (roughly) the acquisition of a grammar;
>
> Third, to aim for a minimally psychologically plausible acquisition procedure, one that relies only upon the evidence known to be available to children for syntactic acquisition.

This chapter presents the details of a LISP-implemented acquisition algorithm tailored after this approach. To illustrate the algorithm the simple Verb Phrase base rule and auxiliary verb inversion scenarios covered in Chapter One will be covered in detail. Recall that the procedure acquires its new rules by attempting to parse presented sentences. The system will acquire its first base rules by working its way through the sentence, *Sue did kiss Mitch.* About a dozen base rules will be acquired along the way. After acquiring the rules to handle this sentence, in the second scenario the procedure will be presented with the auxiliary-inverted form of the same declarative, i.e., *Did Sue kiss Mitch*? Here, a single new *switch* rule will suffice to convert the question into a form parsable by the grammar rules already acquired in the first scenario.

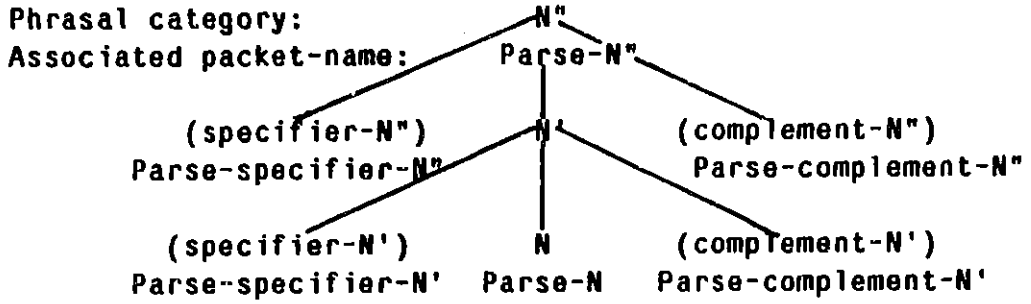
3.2 The Declarative Sentence Scenario

To begin, suppose that the LPARSIFAL system starts its life with no grammar rules and just the two bare X-bar schemas for nouns and verbs, as well as some way to label each lexical item as *noun*, *verb*, or *other*.

More precisely, this research shall adopt a version of Jackendoff's X-bar system, and presume that the initially provided base schema rules are:[1]



And as stated in Chapter One, we associate packet-names with each component of the two schemas:



As discussed earlier, this set of packet-names would be activated and deactivated in a control sequence determined by the structure of the tree above. That is, the typical flow of packet activation is:

Parse-specifier-N"⇒Parse-specifier-N'⇒Parse-N⇒
Parse-complement-N'⇒Parse-complement-N"

---

1. The choice of two levels of "X" structure for the schemas can be justified on the basis of empirical data drawn from English. The supporting arguments for this claim will not be covered in this report; see Jackendoff [1977] for further discussion.

Since specifiers and complements are optional, a parse is permitted to advance to a new packet if no grammar rule can be built to deal with a potential specifier or complement; this feature will prove to be important in the scenarios to come.

Converting this system to LISP is straightforward. To keep track of the current part of the schema under expansion, the variable **Ps-pntr** is set to the name corresponding to the appropriate part of the X-bar schema. Since the original PARSIFAL had to suspend processing certain phrases (like sentences) while it parsed other phrases (for example, Noun Phrases), this method actually requires a stack of **Ps-pntrs**, one pointer for each pending X" under active construction.

The data structure that the pointer steps along is likewise a straightforward mirroring of the X-bar tree structure. The added twist is that in order to capture the evident nesting between the X" and X' levels, one must add a corresponding bracketing in the list structure:

```
Parse-X" schema: (Parse-specifier-X  (Parse-X') Parse-complement-X")
Parse-X' schema: (Parse-specifier-X'  Parse-X   Parse-complement-X')
```

Let us step through this structure to see how the list structure mimics that of the X-bar tree.

Recall first that PARSIFAL already creates certain nodes on demand -- Noun Phrase nodes are created when the parser detects a "leading edge" of a Noun Phrase in the buffer. However, this was done via the execution of grammar rules. We can replace this function of grammar rules by utilizing the the phrase structure schemas in exactly the same way. That is, whenever an "X" schema is entered (as sparked by the "leading edge" for a category of type X), a node of type X" is created and pushed onto the current active node stack as the current active node. At the same time, **Ps-pntr** is set to the first item of the Parse-X" schema list, **Parse-specifier-X"**, activating any grammar rules associated with this packet. The Parse-specifier-X" portion will eventually be processed, with either some tokens in the buffer successfully attached as the specifier of X" or else the X" specifier will be left empty (the attachment of course is via the triggering of associated grammar rules). Because specifiers are optional, the creation of the Specifier-X' attachment point itself is done purely on demand: when a token is about to be attached to the X" as a specifier, it is joined via an intermediate Specifier-X" node. If no X" specifier is discovered, then no such specifier node will be created (the node will be non-existent in the final parse tree). Either way, an *attach* or the failure to trigger any know grammar rules will increment **Ps-pntr**, setting it to the next portion of the Parse-X" schema, **(Parse-X')**.

At this point, the next possible attachment must be an X' specifier to an X' node. Since the X' node is obligatory in any case, the proper step is to automatically create an X' node and set it as the current active node. This is done by exploiting the already-encoded bracketing around **Parse-X'** as a trigger:

when Ps-pntr is advanced so as to encounter the left parenthesis around Parse-X', an X' node is automatically created and pushed onto the active node stack. (In turn this shoves the current active node, X", one level down. The X" becomes the current cyclic node above the current active node, and so is still available as a trigger for rule pattern matching.)

Grammar rules associated with the first packet of the X' schema, Parse-specifier-X' are now free to do their work, possibly attaching an X' specifier to the X'. In turn, the main lexical item X and a (possible) complement to the X' node are dealt with. As soon as the X' complement parse is completed, the construction of the X' node must likewise be at an end. Now LPARSIFAL can rely on the automatic procedures of the original PARSIFAL. Detecting the end of the X' sub-schema (as indicated by the closing right parenthesis in the schema data structure), the completed but as yet unattached X' node can be dropped into the first buffer position, and then promptly attached by a grammar rule to the X" node. The active node stack will be popped, revealing X" once again as the current active node, and the Ps-Pntr can be advanced to the Parse-Complement-X" portion of the base schema. Provided that phrase structure schemas can be properly triggered, this method provides just the right set of phrase structure nodes so that grammar rules can operate.[2]

A final word about the distinction between the X-bar and traditional phrase structure category labels. It may have been noticed that the X-bar system has no distinguished "S"(entence) category label. Instead, the node V" corresponds to the S. This move is not without theoretical ramifications (it has been disputed in the linguistic literature). A full analysis cannot be presented here; however, some justification for eliminating the special "S" node will be presented in the scenario immediately to come. In any case, the correspondence between X-bar and the more traditional notation is as follows:

X-bar notation	Traditional notation
V"	S
V'	VP (Verb Phrase)
N"	NP (Noun Phrase)

With this preamble about the X-bar system implementation out of the way, suppose then that the system is given the following sentence to parse:

---

2. The original Shipman-Marcus packet--phrase structure scheme also employed explict flags to mark the optionality or obligatoriness of particular phrase structure components. In contrast, the X-bar method eliminates the need for this information, since it is already encoded as part of the category name itself. (Specifiers and complements are optional.) In addition, a third flag that Shipman used to indicate the possible arbitary repetition of certain phrase structure elements (e.g., adjectives) could probably be eliminated as part of the distinctive feature system of the X-bar theory.

Sue did kiss Mitch.

Let us trace through the actions of LPARSIFAL as it works its way left-to-right through this sentence. Before plunging ahead, it might be best to lay out just what rules LPARSIFAL will acquire in interpeting this simple example sentence. Observe first that the sentence is in a canonical (for English) Subject-Verb-Object order. No constituent movements are in sight. The grammar rules that LPARSIFAL acquires must reflect this simplicity; it should not acquire *switches* or *insert traces* that mirror constituent movements. In fact, the procedure will have to acquire about eleven rules in handling this initial sentence, as listed on the next page.

The goal of the scenario is to show how these rules and no others will be acquired:

1&2. Noun attach rule #1:
Attach *Sue* as a Noun to an N', and then N' to the N" (Noun Phrase).

3. Subject attach rule:
Attach the Noun Phrase to the V".

4. Verb attach rule #1:
Attach the verb *did* to the Verb Phrase (V').

5. Verb attach rule #2:
Attach the verb *kissed* to a V'; generalize with Verb attach rule #1.

6&7. Noun Phrase rule #2:
Attach *Mitch* as a Noun to an N'; attach the N' to the N"; generalize with Noun rule #1.

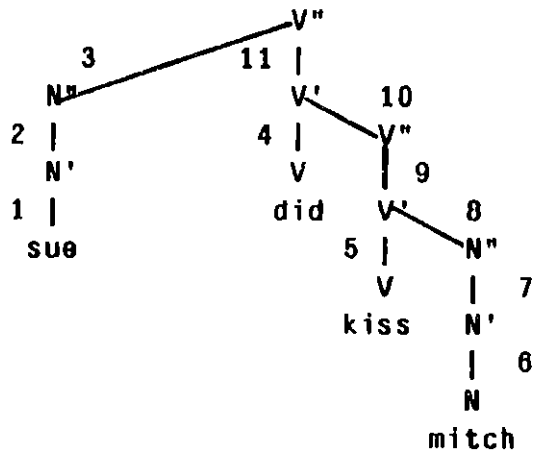
8. Object attach rule:
Attach the N" (*Mitch*) to the V' complement.

9. V' attach rule:
Attach the V' (*kissed Mitch*) to the V".

10. V" attach rule:
Attach the V" (*kissed Mitch*) to the V' complement (*did*)

11. V' attach rule #2:
Attach the V' (*did kiss Mitch*) to the V"; generalize with V' rule #1.

The numbering scheme in the list above refers to the order in which constituents will be attached in the parse tree to be built for this sentence:



Given this sentence, the parser's active node stack and buffer are initially empty:

```
>Sue kissed Mitch.

   The Active Node Stack

C:   NIL NIL /NIL

   The Buffer
     (empty)

Yet unseen words: sue did kiss mitch.

RULES ACQUIRED THIS SESSION:

    (none)
```

Figure 3.1 - The initial stack and buffer state when parsing a declarative.

LPARSIFAL must first engage in a special start-up operation, automatically filling the first buffer position with the first token of the input stream. This procedure deviates from the original

PARSIFAL design, but it is clearly necessary. The original PARSIFAL moved items from the input stream into the buffer on demand -- that is, only when a grammar rule triggering pattern called for a pattern match against the first, second, or third buffer cells. For example, suppose some of the grammar rules in a currently active packet called for pattern matches against the first buffer cell, and other rules demanded matches against the first and second buffer cells. To ajudicate such a match, at least the first two items in the input stream must be pulled into the first two slots of the buffer.

However, since LPARSIFAL currently has no grammar rules, there is nothing to demand that any input tokens to be read into the buffer at all. Without some way of automatically reading in at least one token, the parser will simply stop dead.

The procedure must also create some initial current active node. Otherwise, there will be no node available as an attachment point for items in the buffer. In the original PARSIFAL system, this start-up difficulty was handled by a special initializing rule that created an S(entence) node to prod the parse into motion, and as a side-effect set up that S node as the initial current active node and activated two grammar rule packets as well:

```
{RULE INITIAL-RULE IN NOWHERE
[t] -->
Create a new s node.
!(setq s c).
Activate cpool, ss-start.}
```

This solution seems ad hoc. For one thing, having a special rule to create S-nodes just for the initial setup of a parse is unmotivated (and unnecessary) if in all other cases "ordinary" grammar rules can be used to spark the appropriate creation of S's. As mentioned previously, it is certainly true that S nodes are created all the time in a data-driven fashion; as an example, take any embedded sentence, such as *Sue thought that I was kissing Mitch.* Here, the embedded S *I was kissing Mitch* initiates the creation of an S node -- yet no special rule does the trick. Rather, a data-driven grammar rule detects the tell-tale "leading edge" of an S -- a Noun Phrase followed by a verb -- and so creates a new S node. Listed below are some of Marcus' PARSIFAL rules that dealt with the creation of S nodes in a data-driven fashion:

```
{RULE WH-RELATIVE-CLAUSE IN NP-COMPLETE
[=relpron-np][t] -->
Label c modified.
Attach a new s node labelled sec, relative to c as s.
Activate cpool, parse-subj, wh-pool.

{RULE THAT-S-START PRIORITY: 5 IN CPOOL
[=comp, *that][=np][=verb] -->
Label a new s node sec, comp-s, that-s.
Attach 1st to c as comp.
Attach 2nd to c as np.
Activate cpool, parse-aux.}

{RULE INF-S-START PRIORITY: 5. IN CPOOL
[=*for][=np][=*to] -->
%Handles the marked case; always active%
Label a new s node sec, comp-s, inf-s.
Attach 1st to c as comp.
Attach 2nd to c as np.
Activate cpool, parse-aux.}

{RULE INF-S-START1 PRIORITY: 5. IN INF-COMP
%The COMP can only be dropped if the
complement is expected.%
[=np][=*to,auxverb][=tnsless] -->
Label a new s node sec, comp-s, inf-s.
Attach 1st to c as np.
Activate cpool, parse-aux.}
```

With all these ordinary grammar rules to create S nodes, why should the very first S node created have any different status?

As further support for this position, note that the original PARSIFAL parser created other category nodes -- Noun Phrase nodes -- in exactly the same data driven fashion as the S node creation rules above. Certain "leading edge" triggers for Noun Phrases (e.g., articles such as *the*) prompted the automatic generation of Noun Phrase nodes -- just as in the S node case. What seems to be going on is that when enough of the X-bar structure (for a particular category X) has been encountered to unambiguously determine which X category has been entered, a node is created of that category type. Here, for example, is a PARSIFAL rule that triggered the creation of noun phrases; it utilized a

special feature marker on words, *Ngstart*, to flag items such as articles (*the, an...*) that trigger the start of a Noun Phrase:

```
{AS RULE STARTNP IN CPOOL
[=ngstart] -->
Create a new np node.
If 1st is det then activate parse-det
   else activate parse-qp-1.
Activate npool.}
```

```
Note: "np"= Noun Phrase; "det"=Determiner;
"qp"= Quantifier Phrase (e.g., all people); "npool"= Noun Phrase packet
```

If one adopts this data-driven creation of nodes as the normal way of life for LPARSIFAL, then no initializing rule is needed. Instead, the program simply creates X" nodes of the proper type as they show themselves (unambiguously) in the input stream. Just two such triggering rules have to be assumed. The first is just a re-statement of the X-bar convention; the second is probably derivable from some other assumptions about the nature of predicate-argument structure (the syntax of "logical form"), although this demonstration will not be given here. The two rules are:

> 1. **X" creation:** If a token in the input stream is of type X", and the currently active node is not of type X", create a node of type X" and set it as the current active node.
>
> 2. **Predicate creation:** If the first item in the buffer is of type N" and the second is of type V" (a predicate), then create a V" node and set it as the current active node.

Note that the second rule says that if a Noun Phrase-Verb combination is detected, it must signal the presence of an S node (a V" in the X-bar system adopted here). Likewise, the first rule states that article or a noun must trigger the creation of an N" node.

Returning then to the main story, the automatic start-up action fills the first buffer position with the token *Sue*. By assumption, the program can classify *Sue* as +Noun, more particularly as a *name*.[3] With an item unambiguously of the category +Noun in the first buffer cell, the (known) X-bar schema for N" is triggered. LPARSIFAL automatically creates an N" node, pushes this node onto the active node stack as the current active node (since the stack was empty this has no other visible

3. Recall that some other cognitive machinery accomplishes this, perhaps by noting that *Sue* is a "thing" -- an object.

effect), and activates the corresponding initial packet in the phrase structure schema for N", Parse-specifier-N". Note that this series of actions is equivalent to those that the original PARSIFAL would perform upon detecting a Noun Phrase "leading edge." It leaves the parser in the state shown below.

```
        The Active Node Stack
C:    N"2 (N")/ (PARSE-SPECIFIER-N")

        The Buffer
        ====
1:    WORD11 (*SUE +NOUN NS N3P PROPNOUN NAME) : (sue)


Yet unseen words: did kiss mitch.
```

Figure 3.2 - Starting to construct a Noun Phrase.

Note however that where the original PARSIFAL would execute two grammar rules -- Initial-rule to setup the parse and StartNP to initiate an Noun Phrase "attention shift" -- LPARSIFAL invokes no grammar rules at all to arrive at the corresponding stage of the parse. The X-bar system suffices.

Now that a grammar rule packet has been activated, LPARSIFAL can attempt to match any grammar rules in that packet (Parse-specifier-N") against the features of *Sue* in the first buffer cell. (These features are listed to the right of the lexical items in the figure above.) But since there are no grammar rules in the packet Parse-specifier-N" (there are no grammar rules known at all), no rules can successfully match against the buffer. LPARSIFAL must enter its acquisition phase.

Recall that LPARSIFAL's acquisition procedure is to simply attempt each possible grammar rule action (subject to certain conditions), and save the first one that works. LPARSIFAL cannot do more than this, nor can it invoke the acquisition procedure a second time; if no determination can be made on the basis of current rules and items currently in the buffer, the acquisition procedure simply fails.

One other fact is important for the current example. LPARSIFAL is attempting to build a new grammar rule for a specifier packet, and specifiers are by definition optional. So, if all attempted fixes fail (as they will in this case), LPARSIFAL will simply advance to the next possible packet, Parse-N'.

Following its acquisition procedure, LPARSIFAL first tries to *attach* the item in the first buffer cell, *Sue*, to the current active node, the N", as the specifier of the N". It must therefore evaluate this

potential *attach*. But *Sue* is marked as a *name* -- a definitive N" -- and by the X-bar Convention, cannot go in the specifier slot of the N". (This would violate the X-bar convention that an N" cannot go underneath an N".)

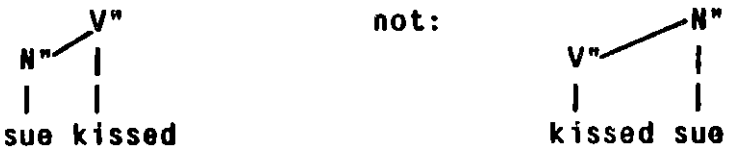
A potential *switch* is next up for consideration. Since there is as yet no item in the second buffer position to even *switch*, at the minimum LPARSIFAL must pull at least the next item of the input stream, *did*, into the buffer. (In fact, an attempted *switch* must always demand the filling of the second buffer cell if it is not already occupied.) Recall however that if a *switch* is attempted LPARSIFAL must often do more work than just this minimal filling of the second buffer slot. In particular, the item just entered into the buffer might itself be the "leading edge" of a whole new constituent. This is true of *did* -- assuming it is marked + V (for verb), the token is really the start of the verb phrase *did kiss Mitch*, and should therefore prompt the creation of a new V". As noted earlier in Chapter One, this could cause problems because *switch* must flip entire constituents, not just single tokens, and LPARSIFAL doesn't know at this point how far the constituent that starts with *did* extends.

In order to deal with this problem, LPARSIFAL's next move would normally be to completely parse the V" constituent that starts with the leading edge *did*. That is, LPARSIFAL would would enter the V" schema, and begin testing grammar rules of the **Parse-specifier-V"** packet (if any) for matches against the buffer. But LPARSIFAL's evaluation of the *switch* via construction of the V" would immediately run aground. Why? To build any part of the V", LPARSIFAL must at least *attach* the verb *did* as the backbone of the V" schema, and then (perhaps) *kiss* and *Mitch*. Since there are no existing grammar rules to do the job, and since LPARSIFAL cannot re-enter its acquisition procedure, the construction of the V", hence the entire *switch*, must fail.[4] The construction of the V" beginning with *switch* is hence out on at least one count.

However, in this case there is another, more important reason why the *switch* is blocked. *Kissed* initiates a V" constituent -- that much is certain. By the locality constraint on *switch*, interchange is possible only if the flip *preserves local syntactic domains*, according to known X-bar constraints. This constraint blocks the *switch*. Why? The V", by definition of predicate-argument structure, *constituent-commands* (**c-commands**) any N"s in the parse tree; N" cannot c-command V".

4. As shown below in Section 3.3, the prohibition on re-entry into the acquisition procedure may have to be weakened slightly for such attention-shifts; one may have to allow the acquisition procedure to be entered one more time. Fortunately, since this *switch* is invalid for another reason, this modification has no effect on the current analysis.

This is because the rules for phrase structure specify that N"s cannot directly dominate V"s:



The success criterion for *switch* is that promptly after the V" is swapped into the first buffer slot, it should be attachable to the N". This would violate the primary rule of constituent-command of N"s by V"s. A V" simply cannot be moved by *switch* into an N" domain.[5]

Because the c-command test is a local check on the application of *switch* -- it proceeds without constructing new material -- it is a test that LPARSIFAL can perform before it engages in a possibly redundant forward check. Faced with *Sue* in the first buffer slot and *did* in the second, LPARSIFAL can thus eliminate a candidate *switch* before ever trying to build a complete V".

Finally, LPARSIFAL will try an *insert trace*. But since a trace too is a sort of name -- a bare N" -- it too cannot be attached as a specifier of N".

In short, all attempted actions have failed. Because the currently active packet is optional, all is not lost; LPARSIFAL advances setting the current value of Ps-pntr to the next packet in the N" schema, Parse-N'.

On entry into the N' portion of the X-bar schema, LPARSIFAL has some additional housekeeping chores. Recall that although the N' specifier may be optional, the N' is not. The procedure's first action is therefore to create an N' node to serve as an attachment point for a possible specfier and an obligatory N. N' is pushed onto the active node stack as the current active node, displacing the N". The N" becomes the cyclic node above the current active node; as such, it is still accessible to rule patterns and actions. Finally, the first part of the N' sub-schema, Parse-specifier-N', is activated.

Unfortunately, once again there are no grammar rules in the currently pointed at packet, Parse-specifier-N'. LPARSIFAL enters its acquisition phase. Since the syntactic relationships

---

5. Note that this restriction would not prohibit such sequences (in English) as *boiled chicken*, where the V" precedes the N" *in surface structure*. It just bans the use of *switch* to arrive at such a sequence. This use of +V items in non-V" phrase structures (here, adjectival positions) is a thesis topic in its own right. It might still be possible to attach the V" *boiled* as part of the N", if there is a way to "de-verbalize" the V", converting it into a category marked +N. These category conversion rules (so-called because they convert an item of one X-bar category into another) have systematicities of their own (see Jackendoff, 1977). Work is underway to integrate theory of such rules with a theory of lexical ambiguity.

between all specifier items and head X-bar items is approximately the same, one would expect that the acquisition attempt with packet **Parse-specifier-N'** should fail just as for **Parse-specifier-N"**; *Sue* cannot be attached as a specifier, the *switch* fails, and so on.[6]

The failure of the acquisition procedure is once again not fatal, because the specifier is optional. LPARSIFAL can advance; the procedure moves on to next packet, **Parse-N**, and activates it. As with the previous packets, there are no grammar rules currently in packet Parse-N, and LPARSIFAL must turn to its acquisition procedure for help.

Here LPARSIFAL must first determine whether an *attach* of *Sue* to the N' portion of the N" X-bar skeleton is possible. By the X-bar convention it is; the lexical item *Sue* is known as +N, and so can be attached to the N'. Since the *attach* succeeds (the first right thing LPARSIFAL has done), the action is saved permanently by storing the pattern of the buffer and active node stack as a new grammar rule trigger, and *attach* as the new rule action. This is LPARSIFAL's very first acquired grammar rule. Below are shown a trace of the acquisition of this rule, followed by a snapshot of the parser state as this new attach rule is about to be executed.

---

6. This is not quite true. In particular, it glosses over one remaining difficulty in handling the initial sentence. Why can't a known name like *Sue* be attached as the N' specifier? This would not be a direct violation of the X-bar constraints: consider noun-noun combinations in English, where one noun obviously is a specifier of the other, such as *baby doll, garden path*. But clearly the noun-noun case doesn't apply with the example sentence, since the first item is the noun *Sue* and the second is the verb *kissed*. One possible fix might be to let the procedure attempt to attach *Sue* as a specifier, and then discover either that the next item is not a noun, or else that it cannot built a valid predicate-argument structure, because it will "use up" the verb to fill a noun slot.

```
GIVE ME A NAME FOR A RULE
BEING CREATED IN PACKET PARSE-N
pattern of rule is:

CYCLIC NODE:  (N")
           C:  (N')
         1ST:  (*SUE NGSTART +NOUN NS N3P PROPNOUN NAME)
         2ND:  (*DID +VERB +TENSE PAST VSPL)
         3RD:   empty

>attach-noun1

  --> About to run: RULE1 ( ATTACH-NOUN1 )

          The Active Node Stack
          N"2 (N")/ (PARSE-N')

C:        N'1 (N')/ (PARSE-N)

          The Buffer
         ====
1:        WORD11 (*SUE NGSTART +NOUN NS N3P ...) : (sue)
2:        WORD12 (*DID +VERB +TENSE PAST VSPL) : (did)

Yet unseen words: kiss mitch.

RULES ACQUIRED THIS SESSION:
  RULE1 ATTACH-NOUN1
```

Figure 3.3 - About to run the noun-building rule.

As usual, the attachment of the first buffer cell item (*Sue*) to the current active node automatically moves *did* into the first buffer cell slot.

Having successfully attached an item to the N' of the N" schema, LPARSIFAL now steps past the currently active packet Parse-N, turning it off, and moves on to activate the next possible packet, Parse-complement-N'. Since there are no grammar rules currently in this packet, LPARSIFAL must try its hand at acquisition.

Deploying its by now familiar and tedious methods, LPARSIFAL first attempts to *attach* the first item in the buffer (*did*) as the complement of the N'. Since *did* is known by assumption to be marked +V, the c-command constraint on possible phrase structure constructions again comes into play. Bare V"s cannot be (directly) attached to Ns; the *attach* fails.

What about a *switch* or a *trace*? The first action would leave us in rather the same position as before the interchange, with the verb *kissed* in the first buffer position, and no way to *attach* the verb. A *trace* would result in direct attachment of an N" to the N", and so is ruled out. Fortunately, the currently active packet Parse-complement-N' is optional. LPARSIFAL advances its phrase structure pointer, and promptly runs into the end of the N' sub-schema:

```
(Parse-specifier-N' Parse-N Parse-complement-N')  ?
                                                  ↑
```

As always, the current active node and all its associated structure -- the N' with the attached noun *Sue* -- is automatically dropped into the first buffer cell, restoring N" as the current active node, and Parse-N' as the active packet. With no further X-bar actions to perform, and with no grammar rules in packet Parse-N', LPARSIFAL will attempt to acquire a grammar rule. Only a synopsis of this particular acquisition will be presented. First, LPARSIFAL will try to *attach* the N' in the first buffer cell to the N". This action meets all X-bar conditions for well-formedness, so the procedure will in fact construct such a rule (RULE2), and *attach* the N' to the N", *did* sliding over once again to take up the buffer slot left behind.

Having attached an item, the procedure advances the Ps-pntr to the next X-bar component down the line, Parse-complement-N". Here matters proceed roughly as they did when Parse-complement-N' was the active packet; the X-bar checks will thwart all attempts at a successful attachment of *did* (as

the reader may verify). LPARSIFAL marches onwards to the next portion of the N" schema:

```
(Parse-specifier-N" Parse-N' Parse-complement-N")    ?
                                                     ↑
```

However, there is no next part to the N" schema; LPARSIFAL discovers that it has run off the end of the N" packet list. The program responds to the end-of-list indication by assuming that the X" item it was constructing is complete, and as usual it drops all completed (but unattached) nodes like this one into the first buffer position, sliding the word *did* previously in the first position over to the right by one:

```
                The Active Node Stack
                NIL

                The Buffer
                ==========
1:              N"2 (N"2 +NOUN NAME NS N3P NOT-MODIFIABLE) : (sue)
2:              WORD12 (*DID +VERB +TENSE PAST VSPL) : (did)

Yet unseen words: kiss mitch.
```

Figure 3.4 - After the N" is dropped into the buffer.

If nothing further could transpire, LPARSIFAL would now be stymied. There are no active packets, and so no way to even see whether a known grammar rule might work. However, by the predicate formation rule, the presence of an N" in the first buffer position and a +V item in the second (*did*) announces the existence of a V". LPARSIFAL responds by creating a new V" node and making it the current active node; the first packet associated with the V" schema, Parse-specifier-V", becomes the currently active packet.

With an active packet and an active node, LPARSIFAL is back in business. Not for long, however; there are no known grammar rules in the Parse-specifier-V" packet. LPARSIFAL must enter its acquisition phase yet again. As its first attempted remedial move, LPARSIFAL tries to see whether the first buffer cell item, N", is permitted as the V" specifier. Since N" is indeed a permitted daughter of V", the attach check succeeds. LPARSIFAL saves the resulting rule. Note that this rule is the (acquired) equivalent of an "unmarked-order" rule for normal English declarative sentences -- one of the sought-after "canonical order" rules.

```
GIVE ME A NAME FOR A RULE
BEING CREATED IN PACKET PARSE-SPECIFIER-V"
pattern of rule is:

CYCLIC NODE: NIL
          C:  (V")
        1ST:  (N" NAME NS N3P NOT-MODIFIABLE)
        2ND:  (*DID +VERB +TENSE PAST VSPL)
        3RD:  empty

action of rule is:
 ATTACH

>unmarked-order

 --> About to run: RULE3 ( UNMARKED-ORDER )

          The Active Node Stack
C:         V"2 (V") / (PARSE-SPECIFIER-V")

          The Buffer
         ====
1:         N"2 (N" NAME NS N3P NOT-MODIFIABLE) : (sue)
2:         WORD12 (*DID +VERB +TENSE PAST VSPL) : (did)

RULES ACQUIRED THIS SESSION:
  RULE3 UNMARKED-ORDER
  RULE2 N'-ATTACH1
  RULE1 ATTACH-NOUN1
```

Figure 3.5 - About to attach the N" to the V" node.

With the attachment of the N", LPARSIFAL deactivates the Parse-specifier-V" packet and enters the sub-schema for V'. The usual actions ensue: V' is made the current active node; V" becomes the current cyclic node; and the packet Parse-specifier-V' is activated. Finally, *did* slides left to take up the spot vacated by the attached N".

Since there are no grammar rules in the Specifier-V' packet, LPARSIFAL starts testing candidate actions. First, can an *attach* of the +V item *did* to the V' succeed? No: by the X-bar rules, a +V (lexical) item fits only under the V slot of the bar system, not the V' portion (the V' specifier or complement). Next, a *switch* must be evaluated. LPARSIFAL notes that the second buffer slot is currently empty, so in preparation for an exchange it pulls in the next token of the input stream, *kiss*,

and fills the second buffer cell. Note however that *kiss* is marked +V, and so is the possible edge of a larger constituent. In order to test the *switch*, the procedure must first attempt to build a complete constituent that starts with *kiss*. Although the first packet activated upon entry into the V" schema, Parse-specifier-V", does have at least one rule in it -- the unmarked-order rule just acquired -- unfortunately it is a rule that attaches N"s to V"s, and consequently has a trigger pattern that demands a N" in the first buffer slot. Since *kiss* is not an N", there are no rules to proceed any further. Thus the potential *switch* cannot be evaluated, and fails.

Finally, a *trace* is up for consideration. It too fails; there is no known grammar rule to attach an N" to the active V' node.[7]

Having exhausted its options for acquisition, the procedure steps past the optional Parse-specifier-V' packet, activating the next packet, Parse-V. Here too there are no known grammar rules to execute, but fortunately, the first acquisition action attempted succeeds: *did* is marked +V, and so can fit under the now available V' slot of the X-bar schema:

```
GIVE ME A NAME FOR RULE
BEING CREATED IN PACKET PARSE-VERB
pattern of rule is:

CYCLIC NODE:  (V")
           C:  (V')
         1ST:  (*DID +VERB +TENSE PAST VSPL)
         2ND:  (*KISS +VERB -TENSE PRES V-3S)
         3RD:  empty

>parse-verb
```

Figure 3.6 - Acquiring the *did* attach rule.

7. In addition, the only known bindee of the trace would be the N" immediately to the left -- *Sue*. Whether this would lead to problems for semantic interpretation remains to be seen.

```
--> About to run: RULE4 ( PARSE-VERB )

          The Active Node Stack
          V"2 (V") / (PARSE-V')
                N" : (Sue)

C:        V'2 (V') / (PARSE-V)

          The Buffer
          ====
1:          WORD12 (*DID +VERB +TENSE PAST VSPL) : (did)
2:          WORD13 (*KISS +VERB -TENSE PRES V3S) : (kiss)
3:

Yet unseen words: mitch.

 RULES ACQUIRED THIS SESSION:
  RULE4 PARSE-VERB
  RULE3 UNMARKED-ORDER
  RULE2 N'-ATTACH1
  RULE1 ATTACH-NOUN1
```

Figure 3.7 - About to attach the verb *did*.

One important point to note about the attachment of *did* is that it prompts the copying of all of its special features -- in particular, +TENSE -- to its mother (or *Head*) node, the V'. This feature, called **Percolation to the Head** (or PTH) can be motivated by the X-bar convention; for a full discussion, see Williams [1979; forthcoming in **Linguistic Inquiry**]. PTH replaces PARSIFAL's method of employing grammar rule actions to *label* of nodes with features. (In fact, the same percolation principle was applied above when the noun *Sue* was attached to the N', and the N' to the N"; the N" received all the special person-number features of the lexical item *Sue*.)

As usual, after performing the *attach*, LPARSIFAL advances its **Ps-pntr** to the next possible packet, **Parse-complement-V'**; V' is still the **current active node**. Finally, with *did* attached, the token in the second buffer cell, *kiss*, slides over to occupy the first buffer slot. At this point the procedure notes that the item now in the first buffer cell in fact triggers the start of a V". Since it is currently not in its acquisition phase, this means that a full attention shift can be performed. That is, LPARSIFAL will temporarily suspend the parse of the current active node, the V', and attempt to parse a complete constituent that begins with the token *kiss*. It actions are analogous to those for an N" parse. It moves the virtual start of buffer window to coincide with the leading edge trigger of the V", then creates a new V" node and pushes it onto the active node stack as the new current active node. Since

the left-most edge of the buffer already coincides with the triggering *kiss*, in this case an actual shift of the left-edge of the buffer itself will not be necessary. However, a new V", V"3, will be created and the V" schema entered a second time.
The parser's state is now as follows:

```
                The Active Node Stack
                V"2 (V") / (PARSE-V')
                      N" : (Sue)
                V'2 (V' +TENSE) / (PARSE-COMPLEMENT-V')
                      V  : (did)

C:              V"3 (V") / (PARSE-SPECIFIER-V")

                The Buffer
               ====
1:              WORD13 (*KISS +VERB -TENSE PRES V3S) : (kiss)
2:
3:

Yet unseen words: mitch.
```

Figure 3.8 - After *did* is attached.

With the active packet set to Parse-specifier-V", the procedure at least has a candidate rule to check against the buffer, for this packet is no longer empty. It contains the newly-acquired unmarked-order rule, RULE3. Once again however, since the item in the first buffer position is a verb, and the pattern of RULE3 calls for an N" in the corresponding spot, RULE3 fails to match and run. Attempting acquisition, the procedure first tries to *attach* the verb, but cannot due to the usual X-bar restriction.

A *switch* is next in line for evaluation. LPARSIFAL notes that hte second buffer slot is currently empty, so it pulls in the next token of the input stream, *Mitch*. Next it must make sure that the item in the second buffer position is a complete constituent. In the case at hand, *Mitch* unambiguously triggers an N", so the procedure must perform another attention shift, temporarily suspending consideration of the V" parse so that it can try to build a complete N" constituent in the second buffer position.

This N" parse proceeds roughly as the with the parse of the initial N" *Sue* -- with of course the exception that LPARSIFAL now has one noun building rule (RULE1) in packet Parse-N. All this is to no avail however; even supposing that the N" were built, after a *switch* no known rule could

trigger to attach the N" to the V". (The only N" attaching rule known requires a NIL cyclic node trigger, and the cyclic node is now set to V"; see the discussion immediately below.) Hence the *switch* is out.

Finally, there is the matter of a *trace.* At first glance, this action seems to pass all tests, for bare N"s (such as a trace) can certainly fit under V"s. In addition, there is in indeed an N" attaching rule already known to the system -- namely, RULE3, the rule that attached the N" *Sue* to the main V". In fact such a result would not be entirely objectionable; interpreting the trace binding by co-indexing with the only N" to its left, *Sue*, the form of the sentence would be roughly, *[Sue did [Sue kiss Mitch]].* Thus the sentence would be structurally analogous to embedded sentences such as, *I thought Sue kissed Mitch.*

The problem of course is the same one that plagued the attempted *switch*: the system cannot yet deal with embedded sentences, as careful study of RULE3 reveals. Consider again the pattern for RULE3:

```
CYCLIC NODE:   NIL
          C:   (V")
        1st:   (N" NAME NS N3P NOT-MODIFIABLE)
        2nd:   (*DID +VERB +TENSE PAST VSPL)
        3rd    empty
```

Ignore for the moment the fact that even the first and second buffer cell patterns for RULE3 do not trigger on the current machine state (and so put aside the question of rule generalization). The key problem is that RULE3's pattern specifies a NIL cyclic node -- a non-embedded sentence. The current predicament consists of a non-NIL cyclic node (V"2), and so RULE3 cannot be applied. At least for now, LPARSIFAL cannot drop a trace.

Advancing then past the optional Parse-specifier-V" packet, the procedure encounters the Parse-V' X-bar component. With going into detail about the process, a V' is created as a new current active node, the Parse-specifier-V' packet is activated, and the acquisition procedure fails in its attempt to build a rule that will execute with this packet. Then, packet Parse-V is triggered. Matters here are more fortunate. The item in the first buffer cell, *kissed*, is marked +V, and so can fit under the currently active part of the X-bar schema. Note that even though LPARSIFAL has a verb-attaching rule in this packet (RULE4, the rule that attached *did* as a verb), it must build a new version of this

rule. This is because the pattern for RULE4 and the current machine state do not match:

```
   RULE4 PATTERN                                MACHINE STATE
CYCLIC NODE:  (V")                                (V")
         C:   (V')                                (V')
       1st:   (*did +verb +tense...)          (*Kiss +verb -tense ...)
       2nd:   (*kiss +verb -tense...)         (*mitch +noun ....      )
       3rd:   empty                                empty

    PACKET:   PARSE-V                              PARSE-V
```

LPARSIFAL constructs a new *attach* rule, with the current machine state as its pattern. However, before storing the newly won rule (RULE5) into the packet Parse-V, LPARSIFAL checks to see whether any other rules in the packet perform the same sort of action (*attach*). RULE4 does. Consequently, LPARSIFAL merges RULE4 and RULE5 into RULE6 by intersecting the feature tests called for in their buffer patterns:

```
GIVE ME A NAME FOR
RULE BEING CREATED IN PACKET PARSE-V

Re-indexing RULE5 in packets (PARSE-V)
Re-indexing RULE4 in packets (PARSE-V)
pattern of (merged) rule is:

CYCLIC NODE: (V")
          C: (V')
        1ST: (+VERB)
        2ND: NIL
        3RD: empty

action of rule is:
   ATTACH

>verb-attach2
```

Figure 3.9 - Generalizing the verb attachment rule.

LPARSIFAL then goes ahead and *attaches* the verb *kiss* to the V', as desired. Ps-pntr is updated and set to the next packet, Parse-complement-V' -- note that we have not yet finished with the parse of the current V' or V". Finally, with *kiss* removed, *Mitch* slides over from its second buffer position to

occupy the first buffer cell.

The presence of the N" triggering item *Mitch* now in the first buffer cell prompts the usual entry into an N" schema; an N" is formed, and made the current active node. Clearly, the parse of this new N" will proceed just as with the parse of the first N". It is left as an exercise to demonstrate that both specifier packets will be skipped, just as before. However, when LPARSIFAL finally activates the packet Parse-N, it will encounter a novel condition: the packet is not empty, but now holds the noun attachment rule RULE1. LPARSIFAL must test to see whether this known rule matches against the current buffer and active node features. Unfortunately, it does not: the item in the first buffer cell, *Mitch*, matches well enough (ignoring for the moment the issue of whether the presence of the specific token **sue* in RULE1's pattern also blocks a match), but the second item in the buffer is the final punctuation mark "." (forced in from the input stream as demanded by the pattern match). Comparing the pattern of RULE1 against the machine state we have:

	RULE1 PATTERN	MACHINE STATE
CYCLIC NODE:	N"	N"
C:	N'	N'
1st:	(*sue ngstart ...)	(*mitch ngstart ...)
2nd:	(*kiss +verb ...)	(*. finalpunc ...)
3rd:	empty	empty
PACKET:	PARSE-N	PARSE-N

Since RULE1 fails, LPARSIFAL must proceed to acquire a new grammar rule, RULE7. Details aside, it should be clear that an *attach* of *Mitch* to the N' will succeed. However, before storing the newly won RULE7 into the current packet (Parse-N), LPARSIFAL checks to see whether any other rules in the packet perform the same sort of action (*attach*). RULE1 does. Consequently, LPARSIFAL merges RULE1 and RULE7 and then runs the resulting rule, as shown in the figures on the next two pages.

```
GIVE ME A NAME FOR RULE
BEING CREATED IN PACKET PARSE-N

Re-indexing RULE7 in packets (PARSE-N)
Re-indexing RULE1 in packets (PARSE-N)
pattern of (merged) rule is:

 CYCLIC: (N")
      C: (N')
    1ST: (NGSTART +NOUN NS N3P PROPNOUN NAME)
    2ND: NIL
    3RD: NIL

>attach-noun2
```

Figure 3.10 - Generalizing the Noun attach rule.

```
--> About to run: RULE8 ( ATTACH-NOUN2 )

          The Active Node Stack
          V"2 (V") / (PARSE-V')
                N" : (Sue)
          V'2 (V' +TENSE) / (PARSE-COMPLEMENT-V')
                V  : (did)
          V"3 (V") / (PARSE-V')
          V'3 (V') / (PARSE-COMPLEMENT-V')
                V  : (kiss)
          N"3 (N") / (PARSE-N')

C:        N'3 (N') / (PARSE-N)

         The Buffer
         ====
1:        WORD14 (*MITCH NGSTART +NOUN NS N3P ...) : (mitch)
2:        WORD15 (*. FINALPUNC PUNC) : (.)

Yet unseen words:

RULES ACQUIRED THIS SESSION:
  RULE8 ATTACH-NOUN2 (MERGED FROM: RULE7 RULE1)
  RULE7
  RULE6 ATTACH-VERB2 (MERGED FROM: RULE5 RULE4)
  RULE5
  RULE4 PARSE-VERB
  RULE3 UNMARKED-ORDER
  RULE2 N'-ATTACH1
  RULE1 ATTACH-NOUN1
```

Figure 3.11 - About to attach *Mitch*.

After *Mitch* has been attached to the N', Ps-pntr is stepped to the packet Parse-complement-N'. Without going into the by-now familiar process, the acquisition procedure will be invoked and fail, for the item now in the first buffer cell is the terminator punctuation mark ".", which cannot be attached (by fiat) to the N'.

With the failure of any action in packet Parse-complement-N', LPARSIFAL starts to unwind the now deep list of pending, partially parsed items on its active node stack. The end of the schema for the currently active N' prompts LPARSIFAL to drop the N' with the attached token *mitch* into the buffer, popping the active node stack so that N"3 is revealed once more as the current active node, and Parse-N' as the current active packet. The N' now in the first buffer cell is of course attachable to

the N"; a rule is built to handle this situation (RULE9). Since there is already an N'-attaching rule known to the system in this packet -- RULE2 -- RULE9 and RULE2 will be merged into RULE10.[8] With the N' now attached, the X-bar schema pointer is advanced to the packet Parse-complement-N". Here there are no known grammar rules, but an invocation of the acquisition procedure will come up empty handed; once again the punctuation mark will resist all attempts at attachment. As a result, the X-bar schema for the N" will be at an end, and N"3 will be automatically dropped into the first buffer spot. V'3 (with the attached verb *kissed*) and the packet Parse-complement-V') are now active. The active node stack has been unwound somewhat; the parser state is now as shown below.

```
                The Active Node Stack
                V"2 (V") / (PARSE-V')
                      N" : (Sue)
                V'2 (V' +TENSE) / (PARSE-COMPLEMENT-V')
                       : (did)
                V"3 (V") / (PARSE-V')

C:              V'3 (V') / (PARSE-COMPLEMENT-V')
                      V  : (kiss)

              The Buffer
    ====
1:            N"3 (N" +NOUN NS N3P ...) : (mitch)
2:            WORD16 (*. FINALPUNC PUNC) : (.)

Yet unseen words:
```

Figure 3.12 - Parser state after dropping the object N" into the buffer.

The parse is nearly at an end. The acquisition procedure allows an *attach* of the N" (*Mitch*) to the V' node, storing a new RULE11. This new grammar rule is the the equivalent of a rule that attaches canonical-position object Noun Phrases to the complement of a Verb Phrase. (The rule is placed into packet Parse-complement-V'.) With this attachment, the base schema for V'3 is also at an end, so the

8. The details are mostly uninteresting, but the effect of the buffer feature merger will be to generalize the pattern demanded for the second buffer cell from [*Kiss ...] to NIL.

V' is dropped into the first cell of the buffer. The parse has now progressed:

```
        The Active Node Stack
              V"2 (V") / (PARSE-V')
                    N" : (Sue)
              V'2 (V' +TENSE) / (PARSE-COMPLEMENT-V')
                    V  : (did)

C:            (V") / (PARSE-V')

              The Buffer
             ====
1:            V'3 (V' +VERB .....) (kiss mitch)
2:            WORD15 (*. FINALPUNC PUNC) : (.)
```

Figure 3.13 - Parser state after the object N" is attached and the completed V' dropped into the buffer.

The acquisition procedure now allows the *attach* of the V' in the first buffer slot to the currently active V" node; the result is saved as a new rule, RULE12. This completes the construction of the V"3 -- it is now the complete constituent *kiss Mitch* -- and so the V" too is now dropped into the buffer, leaving the current active node as V'2 (the node with *did* currently attached) and Parse-complement-V' as the active packet:

```
              The Active Node Stack
              V"2 (V") / (PARSE-V')
                    N" : (Sue)

C:            V'2 (V' +TENSE) / (PARSE-COMPLEMENT-V')
                    V  : (did)

              The Buffer
             ====
1:            V"3 (V" +VERB .....) (kiss mitch)
2:            WORD15 (*. FINALPUNC PUNC) : (.)
```

Figure 3.14 - Parser state after the V' is attached and the completed V" dropped into the buffer.

Tirelessly, LPARSIFAL will construct a new rule to attach the V" (*kiss Mitch*) as the complement of

V'2, RULE13. This completes the construction of V'2, so the entire sub-tree below V'2 -- the completed Verb Phrase -- is dropped into the buffer. This action at last restores the top-most V" as the current active node:

```
        The Active Node Stack
C:          V"2 (V") / (PARSE-V')
                  N" : (Sue)

            The Buffer
            ==========
1:          V'2 (V' +VERB .....) (did kiss mitch)
2:          WORD15 (*. FINALPUNC PUNC) : (.)
```

Figure 3.15 - Parser state after the V' (verb phrase) is attached and the completed V" dropped into the buffer.

The procedure finishes in a rush, adding a new rule to attach the V' to the top-level V" (a merger of the previous V'-V" attach rule); the features of V', in particular +TENSE, finally percolate (via the PTH convention) to reside in the V" node, as required. The action still leaves the final punctuation mark dangling in the first buffer cell:

```
            The Active Node Stack
C:          V"2 (V"+TENSE) / (PARSE-COMPLEMENT-V")
                        N" : (Sue)
                        V' : (did)
                              V" : (kiss)
                                    N" : (Mitch)

            The Buffer
            ==========
1:          WORD15 (*. FINALPUNC PUNC) : (.)
```

Figure 3.16 - Parser state after the main V" is finished.

There seems to be no way to acquire a rule to attach the final punctuation mark, *unless* one adds a special provision that the punctuation mark can be attached to the V"-complement when the cyclic node above the current active node is NIL. But this is certainly ad hoc; it seems to be just another way of defining the notion of "final punctuation." In any case, this does not seem to be a serious matter. The parse is complete.

How has LPARSIFAL done? Though exhausting, the first sentence prompted the creation of exactly the right grammar rules -- the rules for handling English declarative sentences that follow base-generated (canonical) phrase structure order. No constituent movement rules were acquired. A comparison of the rules acquired during this session and the checklist of eleven rules listed at beginning of this section reveals that LPARSIFAL has indeed acquired all the appropriate rules.[9]

9. In addition, the rule generalizations that were made were satisfactory, though the details cannot be covered here. Finally, the tree that has been just been built is intriguing from a linguistic viewpoint: it corresponds to an analysis of the auxiliary verb system that was initially proposed over ten years ago. The structure so built appears to account for a number of the properties of the English auxiliary verb system, in particular, the ordering of auxiliaries, *do*-support, and auxiliary verb inversion.

3.3 The Auxiliary Verb Inversion Scenario

Having handled its first sentence, suppose now that LPARSIFAL is given the sentence,

Did Sue kiss Mitch?

LPARSIFAL now has most of the rules to parse this sentence -- save for the twist at the start. Consequently, the analysis of the parse for this new example will be carried out only to the point where the new grammar rule to deal with the auxiliary verb inversion is acquired. The rest of the parse proceeds just as though the sentence were a simple declarative, and will not be described here.

The initial state of the parse after the initial token *did* is pulled into the first buffer cell is as below:

```
>Did Sue kiss Mitch?

          The Active Node Stack

C:        NIL NIL /NIL

          The Buffer
          ====
1:        WORD27 (*DO +VERB +TENSE PAST VSPL) : (did)

Yet unseen words:  Sue kiss mitch ?
```

Figure 3.17 - Initial parser state for aux inversion.

With the +V item *did* in the first buffer cell, the V" schema is triggered. A V" node is created and set as the current active node; Parse-specifier-V" becomes the current active packet. Recall that there is one grammar rule in this packet, RULE3. This rule was acquired in the previous example sentence, and was designed to attach a subject N" (a Noun Phrase) to a V". LPARSIFAL must check to see whether its pattern matches against the current machine state:

RULE3 PATTERN		MACHINE STATE
Cyclic node:	NIL	NIL
C:	V"	V"
1st:	N"	*did*
2nd:	*did*	empty

The features of the item currently in the parser's buffer cells do not match the pattern of RULE3. LPARSIFAL therefore enters its acquisition phase. First, it tries to attach *did* to the V" specifier; this is out since *did* is marked +V and so can only fit under the V portion of the X-bar schema.

A *switch* is next up for consideration. Since there is no token currently in the second buffer slot to even perform a *switch*, LPARSIFAL pulls one in: *Sue* is placed into the second buffer cell. As usual, the entry of the item prompts the procedure to check whether this item requires an attention shift so that it may be built as a complete constituent before a *switch*. Because *Sue* is assumed marked as +N (a name), the N" schema is entered:

```
        The Active Node Stack
             V"4 (V") / (PARSE-SPECIFIER-V")

C:           N"6 (N") / (PARSE-SPECIFIER-N")

             The Buffer
1:           WORD27 (*DO +VERB +TENSE PAST VSPL) : (did)
             =====
2:           WORD30 (*SUE NGSTART +NOUN NS N3P ...) : (sue)
3:

Yet unseen words:  kiss mitch ?
```

Figure 3.18 - Attention shift to build the N" *Mitch*.

Although the details of the N" parse will be omitted here, LPARSIFAL clearly has already acquired the grammar rules to handle the N" analysis. In particular, RULE8 and RULE10 will correctly trigger and attach *Sue* as the noun to the N' and then the N' to the N". (The reader may verify that the triggering patterns for both rules require only that an N (respectively, an N') be in the first buffer cell, and any item whatsoever in the second buffer cell.) After the attachment via RULE10 of N' to the N", the N" schema will advance to **Parse-complement-N"**, but since the next item in line, *kiss*, is marked +V, it cannot be attached to the N"; N" construction is at an end.

The N" is thus dropped back into the buffer:[10]

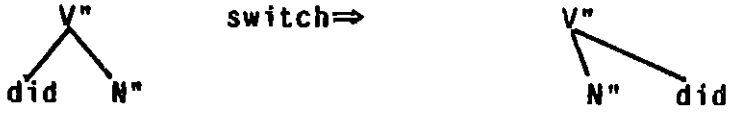
```
The Active Node Stack
      V"4 (V") / (PARSE-SPECIFIER-V")

      The Buffer
      =====
1:    WORD27 (*DO +VERB +TENSE PAST VSPL) : (did)
2:    N"6 (N" NAME NS N3P NOT-MODIFIABLE) : (sue)
3:    WORD31 (*KISS +VERB -TENSE PRES V-3S) : (kiss)

Yet unseen words: mitch ?
```

Figure 3.19 - Parser state after attention shift construction of the N" *sue*.

There is one more condition to meet: the constituent-command constraint on *switch*. The interchange cannot destroy a local syntactic domain. Fortunately, all is well in this case, for the *switch* of the N" into the first buffer cell preserves the domination of N"s by V"s:



10. This analysis again reveals one difficulty. In order to know that *kiss* cannot be so attached, the attention shift must be able to use the X-bar constraints. This appears to be a relaxation of the ban on invoking the acquisition procedure tests more than once for any given try at acquiring a new rule. It may be that attention shifts that start in the second or third buffer cell may themselves be allowed to try the acquisition procedure once, and then no more. This problem is related to the problem of "nested attention shifts" that Marcus encountered. We would like to stipulate some upper limit on the possible nesting, else the look-ahead can probe arbitarily far ahead. This potential unboundedness poses a corresponding problem for acquisition. In any case, knowing just when a constituent is "complete" remains a difficult problem; the methods adopted here do not always work.

At last then the switch can be performed:

```
        The Active Node Stack
              V"4 (V") / (PARSE-SPECIFIER-V")

              The Buffer
             =====
1:            N"6 (N" NAME NS N3P NOT-MODIFIABLE) : (sue)
2:            WORD27 (*DO +VERB +TENSE PAST VSPL) : (did)
3:            WORD31 (*KISS +VERB -TENSE PRES V-3S) : (kiss)

Yet unseen words: mitch ?
```

Figure 3.20 - Parser state after an auxiliary verb switch.

Note that now the buffer and current active node features match the pattern of RULE3 exactly. RULE3 runs, attaching the N" *Sue* to the currently active V" node:

```
GIVE ME A NAME FOR
RULE BEING CREATED IN PACKET PARSE-SPECIFIER-V"

pattern of rule is:

CYCLIC NODE: NIL
          C: (V")
        1ST: (*DO +VERB +TENSE PAST VSPL)
        2ND: (N" NAME NS N3P NOT-MODIFIABLE)
        3RD: (*KISS VERB PRES V-3S TNSLESS)

action of rule is:
SWITCH

>aux-inversion
```

Figure 3.21 - Saving the auxiliary verb inversion rule.

Finally, LPARSIFAL labels the attached node with the rules invoked to build it, in this way recording the fact of auxiliary inversion for any "interpretation" rule that might need to distinguish between an inverted sentence and its canonical order counterpart.

Clearly, the parser now finds itself in exactly the same state as if it had been parsing *Sue did kiss Mitch* all along; an N" is attached as the specifier of a V", and the tokens *did* and *kiss* occupy the first and second buffer cells. Because LPARSIFAL has no memory for what has gone before, the the parse will simply finish without incident, just as if a simple declarative were being analyzed.

This completes LAPRSIFAL's acquisition of its first auxiliary verb inversion sentence. Clearly, other auxiliary verb inverted sentences of the same general form but with different auxiliary verbs will lead to a generalization of the *switch* rule via pattern generalization. For example, if LPARSIFAL is now given the sentence,

```
Could the dog bite Sue?
```

the presence of the modal auxiliary *could* will prompt the creation of a new *switch* rule. Because this *switch* will also be formed in packet Parse-complement-V", its pattern features will be merged with those of the older *switch* to arrive at the following generalized inversion rule:

```
    PACKET: PARSE-SUBJ

Cyclic node: NIL
          C: V"
        1st: (+VERB)
        2nd: (N")
        3rd: (+VERB -TENSE) -->

    SWITCH
```

This form of the auxiliary verb inversion rule is now both general enough to be extended to additional auxiliary verb inversion cases and specific enough not to trigger in the wrong contexts. Significantly, the rule will not run in an embedded sentence. For note that the trigger pattern for the rule demands that the current cyclic node be NIL, and only main Sentences (V"s) will possess this feature.

Since embedded sentences always have a non-NIL cyclic node, the auxiliar verb inversion rule will not trigger on non-grammatical embedded auxiliary verb inversions such as:

```
*I think that did Sue kiss Mitch?
```

This is exactly the desired result. Since LPARSIFAL will never receive a non-grammatical sentence, it will never encounter any evidence that aux-inversion is permitted in embedded sentences, and so will never make an incorrect inductive leap.

To summarize, LPARSIFAL has acquired exactly the right form for an auxiliary verb inversion rule, one that can handle many English auxiliaries. Further, the rule does not trigger exactly when the resulting inversion would be ungrammatical for English speakers, thus also accounting for a class of ungrammatical auxiliary verb inversions. Although the current scenario has illustrated the acquisition of just a single new grammar rule to handle a single sort of deviation from a canonically-ordered sentence, it has highlighted those features of the acquisition procedure that were claimed as crucial for easy acquisition: *local rule construction* and *finite error detectability*. The importance of these constraints holds for the acquisition of a whole variety of other grammar rules, ranging from simple passives to topicalization. All of these rules can be acquired from positive examples using only local refinement, in this manner characterizing the range of syntactic knowledge about parsing that can be easily acquired.

References

Anderson, J. R., (1977) *Induction of Augmented Transition Networks*, Cognitive Science, 1, pp.125-157.

Baker, C. L., (1979) *Syntactic Theory and the Projection Problem*, Linguistic Inquiry, 10, number 4, pp. 533-582.

Berwick R. C., (1980) Computational Complexity, Evaluation Metrics, and Learnability, in preparation.

Biermann, A. and Feldman, J. (1972) *A Survey of Results in Grammatical Inference*, in S. Watanabe, (ed.) Frontiers in Pattern Recognition. New York: Academic Press.

Braine, M. D. S., (1971) *On Two Types of Models of the Internalization of Grammars*, in D.I. Slobin, (ed.), The Ontogenesis of Grammar: A Theoretical Symposium. New York: Academic Press.

Bresnan, J. (1972) Theory of Complementation in English Syntax. Unpublished doctoral dissertation, Department of Linguistics, MIT, Cambridge, MA.

Bresnan, J. (1978) *A Realistic Transformational Grammar*, in M. Halle, J. Bresnan, and G. Miller, (eds.), Linguistic Theory and Psychological Reality. Cambridge, MA: MIT Press.

Bresnan, J. (1980) Polyadicity. Cambridge, MA: Center for Cognitive Studies.

Bresnan, J. and Kaplan, R. (1979) *A Formal System for Grammatical Representation*, in J. Bresnan (ed.), The Mental Representation of Grammatical Relations. Cambridge, MA: MIT Press (to appear).

Brown, R. (1973) A First Language: The Early Stages, Cambridge, MA: Harvard University Press.

Brown, R., and Hanlon, C., (1970) *Derivational Complexity and Order of Acquisition in Child Speech*, in J.R. Hayes, (ed.), Cognition and the Development of Language. New York: John Wiley and Sons.

Carey, S. (1978) *The Child as Word Learner* in M. Halle, J. Bresnan, and G. Miller, (eds.), Linguistic Theory and Psychological Reality. Cambridge MA: MIT Press.

Chomsky, N. (1957) Syntactic Structures. The Hague: Mouton.

Chomsky, N. (1959) *On Certain Formal Properties of Grammars*, in R.D. Luce, R. Bush, and E. Galanter (eds.) Readings in Mathematical Psychology, Vol. 2. New York: Wiley, 1965.

Chomsky, N. (1965) **Aspects of the Theory of Syntax.** Cambridge, MA: MIT Press.

Chomsky, N. (1970) *Remarks on Nominalization*, in R.A. Jacobs and P.S. Rosenbaum, (eds.), **Readings in English Transformational Grammar.** Waltham, MA: Ginn.

Chomsky, N. (1973) *Conditions on Transformations*, in S.R. Anderson and P. Kiparsky, (eds.), **A Festschrift for Morris Halle.** New York: Holt, Rinehart and Winston.

Chomsky, N. (1976) *Conditions on Rules of Grammar*, **Linguistic Analysis, 2,** 303-351.

Chomsky, N. (1980) *On Binding*, **Linguistic Inquiry, 11,** pp. 1-46.

Cross, T. (1977) *Mothers' Speech Adjustments: The Contribution of Selected Child Listener Variables*, in C. Snow and C. Ferguson (eds.), **Talking to Children: Input and Acquisition.** New York: Cambridge University Press.

Culicover, P.W., and Wexler, K. (1977) *Some Syntactic Implications for a Theory of Learnability*, in P.W. Culicover, T. Wasow, and A. Akmajian, (eds.), **Formal Syntax.** New York: Academic Press.

Culicover, P.W., and Wexler, K. (1980) **Formal Principles of Language Acquisition.** Cambridge, MA: MIT Press.

Emonds, J. (1976) **A Transformational Approach to English Syntax.** New York: Academic Press.

Feldman, J. (1972) *Some Decidability Results on Grammatical Inference and Complexity*, **Information and Control, 20,** pp. 244-262.

Fiengo, R. (1974) **Semantic Conditions on Surface Structure**, unpublished doctoral dissertation, Department of Linguistics, MIT, Cambridge, MA.

Fiengo, R. (1977) *On Trace Theory*, **Linguistic Inquiry, 8,** no. 1, pp. 35-61.

Fodor, J. (1968) **Psychological Explanation: An Introduction to the Philosophy of Psychology.** New York: Random House.

Fodor, J.D. (1978) *Parsing Strategies and Constraints on Transformations*, **Linguistic Inquiry, 9,** pp. 427-474.

Frieden, R. (1978) *Cyclicity and the Theory of Grammar*, **Linguistic Inquiry, 9,** pp. 519-550.

Fu, K. and Booth, T. (1975) *Grammatical Inference: Introduction and Survey*, IEEE Transactions Systems, Man, and Cybernetics, SMC-5(1), pp. 95-111; SMC-5(4), pp. 409-423.

Gazdar, G. (1979) Constituent Structures. Sussex: University of Sussex, School of Social Sciences.

Gleitman, L. (1979) Talk at University of California Workshop on Language Learnability, Irvine, CA.

Gold, E.M. (1967) *Language Identification in the Limit*, Information and Control, 10, pp. 447-474.

Goodluck, H., and Solan, L. (1978) (eds.) Papers on the Structure and Development of Child Language, University of Massachusetts Occasional Papers in Linguistics, volume 4. Amherst, MA: University of Massachusetts.

Hamburger H. and Wexler, K. (1975) *A Mathematical Theory of Learning Transformational Grammar*, Journal of Mathematical Psychology, 12, pp. 137-177.

Jackendoff, R. (1977) X-bar Syntax: A Study of Phrase Structure. Cambridge, MA: MIT Press.

Joshi, A.K. and Levy, L.S. (1977) *Constraints on Structural Descriptions: Local Transformations*, SIAM Jounral of Computing, June.

Katz, J. and Fodor, J.A. (1964) *The Structure of a Semantic Theory*, in J.A. Fodor and J.J. Katz,(Eds.) The Structure of Language. Englewood Cliffs, NJ: Prentice-Hall.

Kelley, K. (1967) Early Syntactic Acquisition (Report No. P-3719). Santa Monica, CA: The Rand Corporation.

Kiparsky, P. (1973) *Elsewhere in Phonology*, in S.R. Anderson and P. Kiparsky (eds.), A Festschrift for Morris Halle. New York: Holt, Rinehart, and Winston.

Knobe B. and Knobe, K. (1977) *A Method for Inferring Context-free Grammars*, Information and Control, 31, pp. 129-146.

Koster, J. (1978) Locality Principles in Syntax. Dordrecht, the Netherlands: Foris Publications.

Lasnik, H. and Kupin, J. (1977) *A Restrictive Theory of Transformational Grammar*, Theoretical Linguistics, 4, no. 3, pp. 173-196.

Marcus, M., (1980) A Theory of Syntactic Recognition for Natural Language. Cambridge, MA: MIT Press.

Mayer, J., Erreich, A., and Valian, V. (1978) *Transformations, Basic Operations, and Language Acquisition*, Cognition, 6, pp. 1-14.

McDermott, J. and Forgy, C. (1978) *OPS, A Domain Independent Production System Language*, in D. Waterman and F. Hayes-Roth (eds.) Pattern-Directed Inference Systems. New York: Academic Press.

McNeill, D. (1966) *Developmental Psycholinguistics*, in F. Smith and G. Miller (eds.) The Genesis of Language. Cambridge, MA: MIT Press.

Miller, G., and Chomsky, N. (1963) *Finitary Models of Language Users.* in R.D. Luce, R. Bush, and E. Galanter (eds.) Readings in Mathematical Psychology, Vol. 2. New York: Wiley, 1965.

Newport, E., Gleitman, H., and Gleitman, L. (1977) *Mother, I'd Rather do it Myself: Some Effects and Non-effects of Maternal Speech Style.* in C. Snow and C. Ferguson, Talking to Children: Input and Acquisition, New York: Cambridge University Press.

Oehrle, R. (1974) The Grammatical Status of the English Dative Alternation. Unpublished doctoral dissertation, MIT Department of Linguistics, Cambridge, MA.

Peters, S. and Richie, R. (1973) *On the Generative Power of Transformational Grammar*, Information Science, 6, pp. 49-83.

Pinker, S. (1979) *Formal Models of Language Learning*, Cognition, 7, pp. 217-283.

Pinker, S., Grimshaw, J., and Bresnan, J. (to appear) *Language Acquisition and Lexical Representation* (tentative title), in J. Bresnan (ed.), The Mental Representation of Grammatical Relations. Cambridge, MA: MIT Press.

Putnam, H. (1973) *Philosophy and Our Mental Life*, in H. Putnam, Mind, Language, and Reality, Philosophical Papers Volume II. New York: Cambridge University Press, 1975.

Roeper, T., Bing, J., Lapointe, S., Tavakolian, S. (1979) A Lexical Approach to Language Acquisition, Department of Linguistics, Amherst, MA: University of Massachusetts..

Rychener, M.D., and Newell, A. (1977) *An Instructible Production System: Basic Design Issues*, in D. Waterman and F. Hayes-Roth (eds.) **Pattern-Directed Inference Systems**. New York: Academic Press, 1978.

Shipman, D. (1979) **Phrase Structure Rules for Parsifal**, Working Paper 182. Cambridge, MA: MIT Artificial Intelligence Laboratory.

Siklossy, L. (1972) *Natural Language Learning by Computer*, in H. Simon and L. Siklossy (eds.), **Representation and Meaning: Experiments with Information-Processing Systems**. Englewood Cliffs, NJ: Prentice-Hall.

Snow, C. (1972) *Mothers' Speech to Children Learning Language*, **Child Development**, 43, 549-565.

Solomonoff, R. (1958) *The Mechanization of Linguistic Learning*, **Proceedings of the Second International Congress on Cybernetics**, Belgium, September 3-10, 1958.

Solomonoff, R. (1959) **A New Method for Discovering the Grammars of Phrase Structure Languages**, AFOSR TN 59-110, Washington, D.C.: Office of Scientific Research, United States Air Force, April, 1959.

Thiersch, C. (1978) **Topics in German Syntax**. Unpublished doctoral dissertation, MIT Department of Linguistics. Cambridge, MA.

Wexler, K. (1978) **A Principle Theory for Language Acquisition**. Irvine, CA: University of Social Sciences Research Report number 20.

Wharton, R. (1974) *Approximate Language Identification*,**Information and Control**, 26, pp. 236-255.

Wharton, R. (1977) *Grammar Enumeration and Inference*, **Information and Control**, 33, pp. 253-272.

Williams, E. (1980) *On the Notions "Lexically Related" and Head of a Word*, **Linguistic Inquiry** (to appear).

Winston, P. (1975) *Learning Structural Descriptions from Examples*, in P. Winston (ed.), **The Psychology of Computer Vision**. New York: McGraw-Hill.

Woods W., (1969) *Transition Network Grammars for Natural Language Analysis*, **Communications of the ACM**, 13, pp. 591-606.