A Computational Account of Some Constraints on Language

Mitchell Marcus
MIT Artificial Intelligence Laboratory

In a series of papers over the last several years, Noam Chomsky has argued for several specific properties of language which he claims are universal to all human languages [Chomsky 73, 75, 76]. These properties, which form one of the cornerstones of his current linguistic theory, are embodied in a set of constraints on language, a set of restrictions on the operation of rules of grammar.

This paper will outline two arguments presented at length in [Marcus 77] demonstrating that important subcases of two of these constraints, the Subjacency Principle and the Specified Subject Constraint, fall out naturally from the structure of a grammar interpreter called PARSIFAL, whose structure is in turn based upon the hypothesis that a natural language parser needn't simulate a nondeterministic machine. This "Determinism Hypothesis" claims that natural language can be parsed by a computationally simple mechanism that uses neither backtracking nor pseudo-parallelism, and in which all grammatical structure created by the parser is "indelible" in that it must all be output as part of the structural analysis of the parser's input. Once built, no grammatical structure can be discarded or altered in the course of the parsing process.

In particular, this paper will show that the structure of the grammar interpreter constrains its operation in such a way that, by and large, grammar rules cannot parse sentences which violate either the Specified Subject Constraint or the Subjacency Principle. The component of the grammar interpreter upon which this result principally depends is motivated by the Determinism Hypothesis; this result thus provides indirect evidence for the hypothesis. This result also depends upon the use within a computational framework of the closely related notions of *annotated surface structure* and *trace theory*, which also derive from Chomsky's recent work.

(It should be noted that these constraints are far from universally accepted. They are currently the source of much controversy; for various critiques of Chomsky's position see [Postal 74; Bresnan 76]. However, what is presented below does not argue for these constraints, *per se*, but rather provides a different sort of explanation, based on a processing model, of why the sorts of sentences which these constraints forbid are bad. While the exact formulation of these constraints is controversial, the fact that some set of constraints is needed to account for this range of data is generally agreed upon by most generative grammarians. The account which I will present below *is* crucially linked to Chomsky's, however, in that trace theory is at the heart of this account.)

Because of space limitations, this paper deals only with those grammatical processes characterized by the competence rule "MOVE NP"; the constraints imposed by the grammar interpreter upon those processes characterized by the rule "MOVE WH-phrase" are discussed at length in [Marcus 77] where I show that the behavior characterized by Ross's Complex NP Constraint [Ross 67] itself follows directly from the structure of the grammar interpreter for rather different reasons than the behavior considered in this section. Also because of space limitations, I will not attempt to show that the two constraints I will deal with here *necessarily* follow from the grammar interpreter, but rather only that they *naturally* follow from the interpreter, in particular from a simple, natural formulation of a rule for passivization which itself depends heavily upon the structure of the interpreter. Again, necessity is argued for in detail in [Marcus 77].

This paper will first outline the structure of the grammar interpreter, then present the PASSIVE rule, and then finally show how Chomsky's constraints "fall out" of the formulation of PASSIVE.

Before proceeding with the body of this paper, two other important properties of the parser should be mentioned which will not be discussed here. Both are discussed at length in [Marcus 77]; the first is sketched as well in [Marcus 78]:

1) Simple rules of grammar can by written for this interpreter which elegantly capture the significant generalizations behind not only passivization, but also such constructions as *yes/no* questions, imperatives, and sentences with existential *there*. These rules are reminiscent of the sorts of rules proposed within the framework of the theory of generative grammar, despite the fact that the rules presented here must recover underlying structure given only the terminal string of the surface form of the sentence.

2)The grammar interpreter provides a simple explanation for the difficulty caused by "garden path" sentences, such as "The cotton clothing is made of grows in Mississippi." Rules can be written for this interpreter to

resolve local structural ambiguities which might seem to require nondeterministic parsing; the power of such rules, however, depends upon a parameter of the mechanism. Most structural ambiguities can be resolved, given an appropriate setting of this parameter, but those which typically cause garden paths cannot.

The Structure of PARSIFAL

PARSIFAL maintains two major data structures: a pushdown stack of incomplete constituents called *the active node stack*, and a small three-place *constituent buffer* which contains constituents which are complete, but whose higher level grammatical function is as yet uncertain.

Figure 1 below shows a snapshot of the parser's data structures taken while parsing the sentence "John should have scheduled the meeting.". Note that the active node stack in shown growing *downward*, so that the structure of the stack reflects the structure of the emerging parse tree. At the bottom of the stack is an auxiliary node labelled with the features *modal*, *past*, etc., which has as a daughter the modal "should". Above the bottom of the stack is an S node with an NP as a daughter, dominating the word "John". There are two words in the buffer, the verb "have" in the first buffer cell and the word "scheduled" in the second. The two words "the meeting" have not yet come to the attention of the parser. (The structures of form "(PARSE-AUX CPOOL)" and the like will be explained below.)

The Active Node Stack

S1 (S DECL MAJOR S) / (PARSE-AUX CPOOL)
NP : (John)
AUX1 (MODAL PAST VSPL AUX) / (BUILD-AUX)
MODAL : (should)

The Buffer

1 : WORD3 (*HAVE VERB TNSLESS AUXVERB PRES V-3S) : (have)
2 : WORD4 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ED=EN EN PART PAST ED) : (scheduled)

Yet unseen words: the meeting .

Figure 1 - PARSIFAL's two major data structures.

The constituent buffer is the heart of the grammar interpreter; it is the central feature that distinguishes this parser from all others. The words that make up the parser's input first come to its attention when they appear at the end of this buffer after morphological analysis. Triggered by the words at the beginning of the buffer, the parser may decide to create a new grammatical constituent, create a new node at the bottom of the active node stack, and then begin to attach the constituents in the buffer to it. After this new constituent is completed, the parser will then pop the new constituent from the active node stack; if the grammatical role of this larger structure is as yet undetermined, the parser will insert it into the first cell of the buffer. The parser is free to examine the constituents in the buffer, to act upon them, and to otherwise use the buffer as a workspace.

While the buffer allows the parser to examine some of the context surrounding a given constituent, it does not allow arbitrary look-ahead. The length of the buffer is strictly limited; in the version of the parser presented here, the buffer has only three cells. (The buffer must be extended to five cells to allow the parser to build NPs in a manner which is transparent to the "clause level" grammar rules which will be presented in this paper. This extended parser still has a window of only three cells, but the effective start of the buffer can be changed through an "attention shifting mechanism" whenever the parser is building an NP. In effect, this extended parser has two "logical" buffers of length three, one for NPs and another for clauses, with these two buffers implemented by allowing an overlap in one larger buffer. For details, see [Marcus 77].)

Note that each of the three cells in the buffer can hold a *grammatical constituent* of any type, where a constituent is any tree that the parser has constructed under a single root node. The size of the structure underneath the node is immaterial; both "that" and "that the big green cookie monster's toe got stubbed" are perfectly good constituents once the parser has constructed a subordinate clause from the latter phrase.

The constituent buffer and the active node stack are acted upon by a grammar which is made up of pattern/action rules; this grammar can be viewed as an augmented form of Newell and Simon's production systems [Newell & Simon 72]. Each rule is made up of a pattern, which is matched against some subset of the constituents of the buffer and the accessible nodes in the active node stack (about which more will be said below), and an action, a sequence of operations which acts on these constituents. Each rule is assigned a numerical *priority*, which the grammar interpreter uses to arbitrate simultaneous matches.

The grammar as a whole is structured into *rule packets*, clumps of grammar rules which can be activated and deactivated as a group; the grammar interpreter only attempts to match rules in packets that have been activated by the grammar. Any grammar rule can activate a packet by associating that packet with the constituent at the bottom of the active node stack. As long as that node is at the bottom of the stack, the packets associated with it are active; when that node is pushed into the stack, the packets remain associated with it, but become active again only when that node reaches the bottom of the stack. For example, in figure 1 above, the packet BUILD-AUX is associated with the bottom of the stack, and is thus active, while the packet PARSE-AUX is associated with the S node above the auxiliary.

The grammar rules themselves are written in a language called PIDGIN, an English-like formal language that is translated into LISP by a simple grammar translator based on the notion of top-down operator precedence [Pratt 73]. This use of pseudo-English is similar to the use of pseudo-English in the grammar for Sager's STRING parser [Sager 73]. Figure 2 below gives a schematic overview of the organization of the grammar, and exhibits some of the rules that make up the packet PARSE-AUX.

A few comments on the grammar notation itself are

in order. The general form of each grammar rule is:

{Rule <name> priority: <priority> in <packet>
<pattern> --> <action>}

Each pattern is of the form :

[<description of 1st buffer constituent>] [<2nd>] [<3rd>]

The symbol "=", used only in pattern descriptions, is to be read as "has the feature(s)". Features of the form "*<word>" mean "has the root <word>", e.g. "*have" means "has the root "have"". The tokens "1st", "2nd", "3rd" and "C" (or "c") refer to the constituents in the 1st, 2nd, and 3rd buffer positions and the current active node (i.e. the bottom of the stack), respectively. The PIDGIN code of the rule patterns should otherwise be fairly self-explanatory.

Priority	Pattern Description of: 1st	2nd	3rd	The Stack	Action
			PACKET1		
5:	[ ]	[ ]	[ ]		--> ACTION1
10:	[ ]			[ ]	--> ACTION2
10:	[ ]	[ ]	[ ]	[ ]	--> ACTION3
			PACKET2		
10:	[ ]	[ ]			--> ACTION4
15:	[ ]			[ ]	--> ACTION5

(a) - The structure of the grammar.

{RULE START-AUX PRIORITY: 10. IN PARSE-AUX
[=verb] -->
Create a new aux node.
Label C with the meet of the features of 1st and pres, past, future, tnsless.
Activate build-aux.}

{RULE TO-INFINITIVE PRIORITY: 10. IN PARSE-AUX
[=*to, auxverb] [=tnsless] -->
Label a new aux node inf.
Attach 1st to C as to.
Activate build-aux.}

(b) - Some grammar rules that initiate auxiliaries.

Figure 2

The parser (i.e. the grammar interpreter interpreting some grammar) operates by attaching constituents which are in the buffer to the constituent at the bottom of the stack; functionally, a constituent is in the stack when the parser is attempting to find its daughters, and in the buffer when the parser is attempting to find its mother. Once a constituent in the buffer has been attached, the grammar interpreter will automatically remove it from the buffer, filling in the gap by shifting to the left the constituents formerly to its right. When the parser has completed the constituent at the bottom of the stack, it pops that constituent from the active node stack; the constituent either remains attached to its parent, if it was attached to some larger constituent when it was created, or else it falls into the first cell of the constituent buffer, shifting the buffer to the right to create a gap (and causing an error if the buffer was already full). If the constituents in the buffer provide sufficient evidence that a constituent of a given type should be initiated, a new node of that type can be created and pushed onto the stack; this new node can also be attached to the node at the bottom of the stack before the stack is pushed, if the grammatical function of the new constituent is clear when it is created.

This structure is motivated by several properties which, as is argued in [Marcus 77], any "non-nondeterministic" grammar interpreter must embody. These principles, and their embodiment in PARSIFAL, are as follows:

1) *A deterministic parser must be at least partially data driven.* A grammar for PARSIFAL is made up of pattern/action rules which are triggered when constituents which fulfill specific descriptions appear in the buffer.

2) *A deterministic parser must be able to reflect expectations that follow from the partial structures built up during the parsing process.* Packets of rules can be activated and deactivated by grammar rules to reflect the properties of the constituents in the active node stack.

3) *A deterministic parser must have some sort of constrained look-ahead facility.* PARSIFAL's buffer provides this constrained look-ahead. Because the buffer can hold several constituents, a grammar rule can examine the context that follows the first constituent in the buffer before deciding what grammatical role it fills in a higher level structure. The key idea is that the size of the buffer can be sharply constrained if each location in the buffer can hold a single complete constituent, regardless of that constituent's size. *It must be stressed that this look-ahead ability must be constrained in some manner, as it is here by limiting the length of the buffer; otherwise the "determinism" claim is vacuous.*

The General Grammatical Framework - Traces

The form of the structures that the current grammar builds is based on the notion of *Annotated Surface Structure*. This term has been used in two different senses by Winograd [Winograd 71] and Chomsky [Chomsky 73]; the usage of the term here can be thought of as a synthesis of the two concepts. Following Winograd, this term will be used to refer to a notion of surface structure *annotated by the addition of a set of features* to each node in a parse tree. Following Chomsky, the term will be used to refer to a notion of surface structure annotated by the addition of an element called *trace* to indicate the "underlying position" of "shifted" NPs.

In current linguistic theory, a trace is essentially a "phonologically null" NP in the surface structure representation of a sentence that has no daughters but is "bound" to the NP that filled that position at some level of underlying structure. In a sense, a trace can be viewed as a "dummy" NP that serves as a placeholder for the NP that earlier filled that position; in the same sense, the trace's

binding can be viewed as simply a pointer to that NP. It should be stressed at the outset, however, that a trace is indistinguishable from a normal NP in terms of normal grammatical processes; a trace *is* an NP, even though it is an NP that dominates no lexical material.

There are several reasons for choosing a properly annotated surface structure as a primary output representation for syntactic analysis. While a deeper analysis is needed to recover the predicate/argument structure of a sentence (either in terms of Fillmore case relations [Fillmore 68] or Gruber/Jackendoff "thematic relations" [Gruber 65; Jackendoff 72]), phenomena such as focus, theme, pronominal reference, scope of quantification, and the like can be recovered only from the surface structure of a sentence. By means of proper annotation, it is possible to encode in the surface structure the "deep" syntactic information necessary to recover underlying predicate/argument relations, and thus to encode in the same formalism both deep syntactic relations and the surface order needed for pronominal reference and the other phenomena listed above.

Some examples of the use of trace are given in Figure 3 immediately below.

(1a) What did John give to Sue?
(1b) What did John give *t* to Sue?
|_____|
(1c) John gave *what* to Sue.

(2a) The meeting was scheduled for Wednesday.
(2b) The meeting was scheduled *t* for Wednesday.
|_____|
(2c) ∇ scheduled *a meeting* for Wednesday.

(3a) John was believed to be happy.
(3b) John was believed [$_S$ *t* to be happy].
|_____|

Figure 3 – Some examples of the use of trace.

One use of trace is to indicate the underlying position of the wh-head of a question or relative clause. Thus, the structure built by the parser for 3.1a would include the trace shown in 3.1b, with the trace's binding shown by the line under the sentence. The position of the trace indicates that 3.1a has an underlying structure analogous to the overt surface structure of 3.1c.

Another use of trace is to indicate the underlying position of the surface subject of a passivized clause. For example, 3.2a will be parsed into a structure that includes a trace as shown as 3.2b; this trace indicates that the subject of the passive has the underlying position shown in 3.2c. The symbol "∇" signifies the fact that the subject position of (2c) is filled by an NP that dominates no lexical structure. (Following Chomsky, I assume that a passive sentence in fact has *no underlying subject*, that an agentive "by NP" prepositional phrase originates as such in underlying structure.) The trace in (3b) indicates that the phrase "to be happy", which the brackets show is really an embedded clause, has an underlying subject which is identical with the surface subject of the matrix S, the clause that dominates the embedded complement. Note that what is conceptually the underlying subject of the embedded clause has been passivized into subject position of the matrix S, a phenomenon commonly called "raising". The analysis of this phenomenon assumed here derives from [Chomsky 73]; it is an alternative to the classic analysis which involves "raising" the subject of the embedded clause into object position of the matrix S before passivization (for details of this later analysis see [Postal 74]).

The Passive Rule

In this section and the next, I will briefly sketch a solution to the phenomena of passivization and "raising" in the context of a grammar for PARSIFAL. This section will present the Passive rule; the next section will show how this rule, without alteration, handles the "raising" cases.

Let us begin with the parser in the state shown in figure 4 below, in the midst of parsing 3.2a above. The analysis process for the sentence prior to this point is essentially parallel to the analysis of any simple declarative with one exception: the rule PASSIVE-AUX in packet BUILD-AUX has decoded the passive morphology in the auxiliary and given the auxiliary the feature *passive* (although this feature is not visible in figure 4). At the point we begin our example, the packet SUBJ-VERB is active.

The Active Node Stack (1. deep)
S21 (S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
C: VP17 (VP) / (SUBJ-VERB)
VERB : (scheduled)

The Buffer
1 : PP14 (PP) : (for Wednesday)
2 : WORD162 (*. FINALPUNC PUNC) : (.)

Figure 4 - Partial analysis of a passive sentence: after the verb has been attached.

The packet SUBJ-VERB contains, among other rules, the rule PASSIVE, shown in figure 5 below. The pattern of this rule is fulfilled if the auxiliary of the S node dominating the current active node (which will always be a VP node if packet SUBJ-VERB is active) has the feature *passive*, and the S node has not yet been labelled *np-preposed*. (The notation "** C" indicates that this rule matches against the two accessible nodes in the stack, not against the contents of the buffer.) The action of the rule PASSIVE simply creates a trace, sets the binding of the trace to the subject of the dominating S node, and then drops the new trace into the buffer.

{RULE PASSIVE IN SUBJ-VERB
[** c; the aux of the s above c is passive;
the s above c is not np-preposed] -->
Label the s above c np-preposed.
Create a new np node labelled trace.
Set the binding of c to the np of the s above c.
Drop c.}

Figure 5 - Six lines of code captures np-preposing.

The state of the parser after this rule has been executed, with the parser previously in the state in figure 4 above, is shown in figure 6 below. S21 is now labelled with the feature *np-preposed*, and there is a trace, NP53, in the first buffer position. NP53, as a trace, has no daughters, but is bound to the subject of S21.

The Active Node Stack (1. deep)
S21 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
C: VP17 (VP) / (SUBJ-VERB)
VERB : (scheduled)

The Buffer
1 : NP53 (NP TRACE) : bound to: (The meeting)
2 : PP14 (PP) : (for Wednesday)
3 : WORD162 (*. FINALPUNC PUNC) : (.)

Figure 6 - After PASSIVE has been executed.

Now rules will run which will activate the two packets SS-VP and INF-COMP, given that the verb of VP17 is "schedule". These two packets contain rules for parsing simple objects of non-embedded Ss, and infinitive complements, respectively. Two such rules, each of which utilize an NP immediately following a verb, are given in figure 7 below. The rule OBJECTS, in packet SS-VP, picks up an NP after the verb and attaches it to the VP node as a simple object. The rule INF-S-START1, in packet INF-COMP, triggers when an NP is followed by "to" and a tenseless verb; it initiates an infinitive complement and attaches the NP as its subject. (An example of such a sentence is "We wanted John to give a seminar next week".) The rule INF-S-START1 must have a higher priority than OBJECTS because the pattern of OBJECTS is fulfilled by any situation that fulfills the pattern of INF-S-START1; if both rules are in active packets and match, the higher priority of INF-S-START1 will cause it to be run instead of OBJECTS.

{RULE OBJECTS PRIORITY: 10 IN SS-VP
[=np] -->
Attach 1st to c as np.}

{RULE INF-S-START1 PRIORITY: 5. IN INF-COMP
[=np] [=*to,auxverb] [=tnsless] -->
Label a new s node sec, inf-s.
Attach 1st to c as np.
Activate parse-aux.}

Figure 7 - Two rules which utilize an NP following a verb.

While there is not space to continue the example here in detail, note that the rule OBJECTS will trigger with the parser in the state shown in figure 6 above, and will attach NP53 as the object of the verb "schedule. OBJECTS is thus totally indifferent both to the fact that NP53 was not a regular NP, but rather a trace, and the fact that NP53 did not originate in the input string, but was placed into the buffer by grammatical processes. Whether or not this rule is executed is absolutely unaffected by differences between an active sentence and its passive form; the analysis process for either is identical as of this point in the parsing process. Thus, the analysis process will be exactly parallel in both cases after the PASSIVE rule has been executed. (I remind the reader that the analysis of passive assumed above, following Chomsky, does *not* assume a process of "agent deletion", "subject postposing" or the like.)

Passives in Embedded Complements - "Raising"

The reader may have wondered why PASSIVE drops the trace it creates into the buffer rather than immediately attaching the new trace to the VP node. As we will see below, such a formulation of PASSIVE also correctly analyzes passives like 3.3a above which involve "raising", but with no additional complexity added to the grammar, correctly capturing an important generalization about English. To show the range of the generalization, the example which we will investigate in this section, sentence (1) in figure 8 below, is yet a level more complex than 3.3a above; its analysis is shown schematically in 8.2. In this example there are two traces: the first, the subject of the embedded clause, is bound to the subject of the major clause, the second, the object of the embedded S, is bound to the first trace, and is thus ultimately bound to the subject of the higher S as well. Thus the underlying position of the NP "the meeting" can be viewed as being the object position of the embedded S, as shown in 8.3.

(1)The meeting was believed to have been scheduled for Wednesday.
(2)The meeting was believed [$_S$ *t* to have been scheduled *t* for Wednesday]
(3) ∇ believed [$_S$ ∇ to have scheduled *the meeting* for Wednesday].

Figure 8 - This example shows simple passive and raising.

We begin our example, once again, right after "believed" has been attached to VP20, the current active node, as shown in figure 9 below. Note that the AUX node has been labelled *passive*, although this feature is not shown here.

The Active Node Stack (1. deep)
S22 (S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
C: VP20 (VP) / (SUBJ-VERB)
VERB : (believed)

The Buffer
1 : WORD166 (*TO PREP AUXVERB) : (to)
2 : WORD167 (*HAVE VERB TNSLESS AUXVERB PRES ...) : (have)

Figure 9 - After the verb has been attached.

The packet SUBJ-VERB is now active; the PASSIVE rule, contained in this packet now matches and is executed. This rule, as stated above, creates a trace, binds it to the subject of the current clause, and drops the trace into the first cell in the buffer. The resulting state is shown in figure 10 below.

The Active Node Stack (1. deep)
S22 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
C: VP20 (VP) / (SUBJ-VERB)
VERB : (believed)

The Buffer
1 : NP55 (NP TRACE) : bound to: (The meeting)
2 : WORD166 (*TO PREP AUXVERB) : (to)
3 : WORD167 (*HAVE VERB TNSLESS AUXVERB PRES ...) : (have)

Yet unseen words: been scheduled for Wednesday .

Figure 10 - After PASSIVE has been executed.

Again, rules will now be executed which will activate the packet SS-VP (which contains the rule OBJECTS) and, since "believe" takes infinitive complements, the packet INF-COMP (which contains INF-S-START1), among others. (These rules will also deactivate the packet SUBJ-VERB.) Now the patterns of OBJECTS and INF-S-START1 will both match, and INF-S-START1, shown above in figure 7, will be executed by the interpreter since it has the higher priority. (Note once again that a trace is a perfectly normal NP from the point view of the pattern matching process.) This rule now creates a new S node labelled infinitive and attaches the trace NP55 to the new infinitive as its subject. The resulting state is shown in figure 11 below.

The Active Node Stack (2. deep)
S22 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
VP20 (VP) / (SS-VP THAT-COMP INF-COMP)
VERB : (believed)
C: S23 (SEC INF-S S) / (PARSE-AUX)
NP : bound to: (The meeting)

The Buffer
1 : WORD166 (*TO PREP AUXVERB) : (to)
2 : WORD167 (*HAVE VERB TNSLESS AUXVERB PRES ...) : (have)

Yet unseen words: been scheduled for Wednesday .

Figure 11 - After INF-S-START1 has been executed.

We are now well on our way to the desired analysis. An embedded infinitive has been initiated, and a trace bound to the subject of the dominating S has been attached as its subject, although no rule has explicitly "lowered" the trace from one clause into the other.

The parser will now proceed exactly as in the previous example. It will build the auxiliary, attach it, and attach the verb "scheduled" to a new VP node. Once again PASSIVE will match and be executed, creating a trace, binding it to the subject of the clause (in this case itself a trace), and dropping the new trace into the buffer. Again the rule OBJECTS will attach the trace NP57 as the object of VP21, and the parse will then be completed by grammatical processes which will not be discussed here. An editted form of the tree structure which results is shown in figure 12 below. A trace is indicated in this tree by giving the terminal string of its ultimate binding in parentheses.

```
(NP-PREPOSED S DECL MAJOR)
  NP: (MODIBLE NP DEF DET NP)
        The meeting
  AUX: (PASSIVE PAST V13S AUX)
        was
  VP: (VP)
     VERB: believed
     NP: (NP COMP)
        S: (NP-PREPOSED SEC INF-S S)
           NP: (NP TRACE) (bound* to: The meeting)
           AUX: (PASSIVE PERF INF AUX)
                 to have been
           VP: (VP)
              VERB: scheduled
              NP: (NP TRACE) (bound* to: The meeting)
              PP: (PP)
                 PREP: for
                 NP: (NP TIME DOW)
                       Wednesday
```

Figure 12 - The final tree structure.

This example demonstrates that the simple formulation of the PASSIVE rule presented above, interacting with other simply formulated grammatical rules

for parsing objects and initiating embedded infinitives, allows a trace to be attached either as the object of a verb or as the subject of an embedded infinitive, whichever is the appropriate analysis for a given grammatical situation. Because the PASSIVE rule is formulated in such a way that it drops the trace it creates into the buffer, later rules, already formulated to trigger on an NP in the buffer, will analyze sentences with NP-preposing exactly the same as those without a preposed subject. Thus, we see that the availability of the buffer mechanism is crucial to capturing this generalization; such a generalization can only be stated by a parser with a mechanism much like the buffer used here.

The Grammar Interpreter and Chomsky's Constraints

Before turning now to a sketch of a computational account of Chomsky's constraints, there are several important limitations of this work which must be enumerated.

First of all, while two of Chomsky's constraints seem to fall out of the grammar interpreter, there seems to be no apparent account of a third, the Propositional Island Constraint, in terms of this mechanism.

Second, Chomsky's formulation of these constraints is intended to apply to all rules of grammar, both syntactic rules (i.e. transformations) and those rules of semantic interpretation which Chomsky calls "rules of construal", a set of shallow semantic rules which govern anaphoric processes [Chomsky 77]. The discussion here will only touch on purely syntactic phenomena; the question of how rules of semantic interpretation can be meshed with the framework presented in this document has yet to be investigated.

Third, the arguments presented below deal only with English, and in fact depend strongly upon several facts about English syntax, most crucially upon the fact that English is subject-initial. Whether these arguments can be successfully extended to other language types is an open question, and to this extent this work must be considered exploratory.

And finally, I will not show that these constraints must be true *without exception*; as we will see, there are various situations in which the constraints imposed by the grammar interpreter can be circumvented. Most of these situations, though, will be shown to demand much more complex grammar formulations than those typically needed in the grammar so far constructed. This is quite in keeping with the suggestion made by Chomsky [Chomsky 77] that the constraints are not necessarily without exception, but rather that exceptions will be "highly marked" and therefore will count heavily against any grammar that includes them.

The Specified Subject Constraint

The Specified Subject Constraint (SSC), stated informally, says that no rule may involve two constituents that are Dominated by different cyclic nodes unless the lower of the two is the subject of an S or NP. Thus, no rule may involve constituents X and Y in the structure shown in figure 13 below, if α and β are cyclic nodes and Z is the subject of α, Z distinct from X.

$[_{\beta}...Y...[_{\alpha}\ Z...X...]...Y...]$

Figure 13 - SSC:
No rule can involve X and Y in this structure.

The SSC explains why the surface subject position of verbs like "seems" and "is certain" which have no underlying subject can be filled only by the subject and not the object of the embedded S: The rule "MOVE NP" is free to shift any NP into the empty subject position, but is constrained by the SSC so that the object of the embedded S cannot be moved out of that clause. This explains why (a) in figure 14 below, but not 14b, can be derived from 14c; the derivation of 14b from 14c would violate the SSC.

(a) John seems to like Mary.
(b)*Mary seems John to like.
(c) ∇ seems $[_S$ John to like Mary]

Figure 14 - Some examples illustrating the SSC.

In essence, then, the Specified Subject Constraint constrains the rule "MOVE NP" in such a way that only the subject of a clause can be moved out of that clause into a position in a higher S. Thus, if a trace in an annotated surface structure is bound to an NP Dominated by a higher S, that trace must fill the subject position of the lower clause.

In the remainder of this section I will show that the grammar interpreter constrains grammatical processes in such a way that annotated surface structures constructed by the grammar interpreter will have this same property, given the formulation of the PASSIVE rule presented above. In terms of the parsing process, this means that if a trace is "lowered" from one clause to another as a result of a "MOVE NP"-type operation during the parsing process, then it will be attached as the subject of the second clause. To be more precise, if a trace is attached so that it is Dominated by some S node S1, and the trace is bound to an NP Dominated by some other S node S2, then that trace will necessarily be attached so that it fills the subject position of S1. This is depicted in figure 15 below.

```
        The Active Node Stack
             .....
        S2 ... / ...
             ...
             NP2
             ...
C:      S1 ... / ...
             NP: NP1 (NP TRACE) : bound to NP2
```

Figure 15 - NP1 must be attached as the subject of S1 since it is bound to an NP Dominated by a higher S.

Looking back at the complex passive example involving "raising" presented above, we see that the parsing process results in a structure exactly like that shown above. The original point of the example, of course, was that the rather simple PASSIVE rule handles this case without the need for some mechanism to explicitly lower the NP. The PASSIVE rule captures this generalization by

dropping the trace it creates into the buffer (after appropriately binding the trace), thus allowing other rules written to handle normal NPs (e.g. OBJECTS and INF-S-START1) to correctly place the trace.

This statement of PASSIVE does more, however, than simply capture a generalization about a specific construction. As I will argue in detail below, the behavior specified by both the Specified Subject Constraint and Subjacency follows almost immediately from this formulation. In [Marcus 77], I argue that this formulation of PASSIVE is the only simple, non-*ad hoc*, formulation of this rule possible, and that all other rules characterized by the competence rule "MOVE NP" must operate similarly; here, however, I will only show that these constraints follow naturally from this formulation of PASSIVE, leaving the question of necessity aside. I will also assume one additional constraint below, the *Left-to-Right Constraint*, which will be briefly motivated later in this paper as a natural condition on the formulation of a grammar for this mechanism.

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

I will now show that a trace created by PASSIVE which is bound to an NP in one clause can only serve as the subject of a clause dominated by that first clause.

Given the formulation of PASSIVE, a trace can be "lowered" into one clause from another only by the indirect route of dropping it into the buffer before the subordinate clause node is created, which is exactly how the PASSIVE rule operates. This means that the ordering of the operations is crucially: 1) create a trace and drop it into the buffer, 2) create a subordinate S node, 3) attach the trace to the newly created S node. The key point is that at the time that the subordinate clause node is created and becomes the current active node, the trace must be sitting in the buffer, filling one of the three buffer positions. Thus, the parser will be in the state shown in figure 16 below, with the trace, in fact, most likely in the first buffer position.

The Active Node Stack

.....

C: S123 (S SEC ...) / ...

The Buffer

...

NP123 (NP TRACE) : bound to NP in S above S123

...

Figure 16 - Parser state after embedded S created.

Now, given the L-to-R Constraint, a trace which is in the buffer at the time that an embedded S node is first created must be one of the first several constituents attached to the S node or its daughter nodes. From the structure of English, we know that the leftmost three constituents of an embedded S node, ignoring topicalized constituents, must either be

COMP NP AUX
or
NP AUX [$_{VP}$ VERB ...].

(The COMP node will dominate flags like "that" or "for" that mark the beginning of a complement clause.) But then, if a trace, itself an NP, is one of the first several constituents attached to an embedded clause, the only position it can fill will be the subject of the clause, exactly the empirical consequence of Chomsky's Specified Subject Constraint in such cases as explained above.

The L-to-R Constraint

Let us now return to the motivation for the L-to-R Constraint. Again, I will not attempt to prove that this constraint must be true, but merely to show why it is plausible.

Empirically, the Left-to-Right Constraint seems to hold for the most part; for the grammar of English discussed in this paper, and, it would seem, for any grammar of English that attempts to capture the same range of generalizations as this grammar, the constituents in the buffer are utilized in left-to-right order, with a small range of exceptions. This usage is clearly not enforced by the grammar interpreter as presently implemented; it is quite possible to write a set of grammar rules that specifically ignores a constituent in the buffer until some arbitrary point in the clause, though such a set of rules would be highly *ad hoc*. However, there rarely seems to be a need to remove other than the first constituent in the buffer.

The one exception to the L-to-R Constraint seems to be that a constituent C_i may be attached before the constituent to its left, C_{i-1}, if C_i does not appear in surface structure in its underlying position (or, if one prefers, in its unmarked position) and if its removal from the buffer reestablishes the unmarked order of the remaining constituents, as in the case of the AUX-INVERSION rule discussed earlier in this paper. To capture this notion, the L-to-R Constraint can be restated as follows: All constituents must be attached to higher level constituents according to the left-to-right order of constituents in the unmarked case of that constituent's structure.

This reformulation is interesting in that it would be a natural consequence of the operation of the grammar interpreter if packets were associated with the phrase structure rules of an explicit "base component", and these rules were used as templates to build up the structure assigned by the grammar interpreter. A packet of grammar rules would then be explicitly associated with each symbol on the right hand side of each phrase structure rule. A constituent of a given type would then be constructed by activating the packets associated with each node type of the appropriate phrase structure rule in left-to-right order. Since these base rules would reflect the unmarked l-to-r order of constituents, the constraint suggested here would then simply fall out of the interpreter mechanism.

Subjacency

Before turning to the Subjacency Principle, a few auxiliary technical terms need to be defined: If we can

trace a path up the tree from a given node X to a given node Y, then we say *X is dominated by Y*, or equivalently, *Y dominates X*. If Y dominates X, and no other nodes intervene (i.e. X is a daughter of Y), then *Y immediately (or directly) dominates X*. [Akmajian & Heny 75]. One non-standard definition will prove useful: I will say that if Y dominates X, and Y is a cyclic node, i.e. an S or NP node, and there is no other cyclic node Z such that Y dominates Z and Z dominates X (i.e. there is no intervening cyclic node Z between Y and X) then *Y Dominates X*.

The principle of Subjacency, informally stated, says that no rule can involve constituents that are separated by more than one cyclic node. Let us say that a node X *is subjacent to* a node Y if there is at most one cyclic node, i.e. at most one NP or S node, between the cyclic node that Dominates Y and the node X. Given this definition, the Subjacency principle says that no rule can involve constituents that are not subjacent.

The Subjacency principle implies that movement rules are constrained so that they can move a constituent only into positions that the constituent was subjacent to, i.e. only within the clause (or NP) in which it originates, or into the clause (or NP) that Dominates that clause (...). This means that if α, β, and ϵ in figure 17 are cyclic nodes, no rule can move a constituent from position X to either of the positions Y, where $[_{\alpha}...X...]$ is distinct from $[_{\alpha}X]$.

$[_{\epsilon}...Y...[_{\beta}...[_{\alpha}...X...]...]...Y...]$

Figure 17 - Subjacency:
No rule can involve X and Y in this structure.

Subjacency implies that if a constituent is to be "lifted" up more than one level in constituent structure, this operation must be done by repeated operations. Thus, to use one of Chomsky's examples, the sentence given in figure 18a, with a deep structure analogous to 18b, must be derived as follows (assuming that "is certain", like "seems", has no subject in underlying structure): The deep structure must first undergo a movement operation that results in a structure analogous to 18c, and then another movement operation that results in 18d, each of these movements leaving a trace as shown. That 18c is in fact an intermediate structure is supported by the existence of sentences such as 18e, which purportedly result when the ∇ in the matrix S is replaced by the lexical item "it", and the embedded S is tensed rather than infinitival. The structure given in 18f is ruled out as a possible annotated surface structure, because the single trace could only be left if the NP "John" was moved in one fell swoop from its underlying position to its position in surface structure, which would violate Subjacency.

(a) John seems to be certain to win.
(b) ∇ seems $[_S$ ∇ to be certain $[_S$ John to win]]
(c) ∇ seems $[_S$ John to be certain $[_S$ *t* to win]]
(d) John seems $[_S$ *t* to be certain $[_S$ *t* to win]]
(e) It seems that John is certain to win.
(f) John seems $[_S$ ∇ to be certain $[_S$ *t* to win]]

Figure 18 - An example demonstrating Subjacency.

Having stated Subjacency in terms of the abstract competence theory of generative grammar, I now will show that a parsing correlate of Subjacency follows from the structure of the grammar interpreter. Specifically, I will show that there are only limited cases in which a trace generated by a "MOVE-NP" process can be "lowered" more than one clause, i.e. that a trace created and bound while any given S is current must almost always be attached either to that S or to an S which is Dominated by that S.

Let us begin by examining what it would mean to lower a trace more than one clause. Given that a trace can only be "lowered" by dropping it into the buffer and then creating a subordinate S node, as discussed above, lowering a trace more than one clause necessarily implies the following sequence of events, depicted in figure 19 below: First, a trace NP1 must (a) be created with some S node, S1, as the current S, (b) bound to some NP Dominated by that S and then (c) dropped into the buffer. By definition, it will be inserted into the first cell in the buffer. (This is shown in figure 19a) Then a second S, S2, must be created, supplanting S1 as the current S, and then yet a third S, S3, must be created, becoming the current S. During all these steps, the trace NP1 remains sitting in the buffer. Finally, NP1 is attached under S3 (fig. 19b). By the Specified Subject Constraint, NP1 must then attach to S3 as its subject.

The Active Node Stack
.....
C: S1 ... / ...

The Buffer
1st: NP1 (NP TRACE) : bound to NP Dominated by S1
...

(a) - NP1 is dropped into the buffer
while S1 is the current S.

The Active Node Stack
.....
S1 ... / ...
S2 ... / ...
C: S3 ... / ...
NP1 (NP TRACE) : bound to NP Dominated by S1

(b) - After S2 and S3 are created,
NP1 is attached to S3 as its subject (by the SSC).

Figure 19 - Lowering a trace more than 1 clause

But this sequence of events is highly unlikely. The essence of the argument is this:

Nothing in the buffer can change between the time that S2 is created and S3 is created if NP1 remains in the buffer. NP1, like any other node that is dropped from the active node stack into the buffer, is inserted into the first buffer position. But then, by the L-to-R Constraint, nothing to the right of NP1 can be attached to a higher level constituent until NP1 is attached. (One can show that it is most unlikely that any constituents will enter to the left of NP1 after it is dropped into the buffer, but I will suppress this detail here; the full argument is included in [Marcus 77].)

But if the contents of the buffer do not change between the creation of S2 and S3, then what can possibly motivate the creation of both S2 and S3? The contents of the buffer must necessarily provide clear evidence that both of these clauses are present, since, by the Determinism Hypothesis, the parser must be correct if it initiates a constituent. Thus, the same three constituents in the buffer must provide convincing evidence not only for the creation of S2 but also for S3. Furthermore, if NP1 is to become the subject of S3, and if S2 Dominates S3, then it would seem that the constituents that follow NP1 in the buffer must also be constituents of S3, since S3 must be completed before it is dropped from the active node stack and constituents can then be attached to S2. But then S2 must be created *entirely* on the basis of evidence provided by the constituents of another clause (unless S3 has less than three constituents). Thus, it would seem that the contents of the buffer cannot provide evidence for the presence of both clauses unless the presence of S3, by itself, is enough to provide confirming evidence for the presence of S2. This would be the case only if there were, say, a clausal construction that could only appear (perhaps in a particular environment) as the initial constituent of a higher clause. In this case, if there are such constructions, a violation of Subjacency should be possible.

With the one exception just mentioned, there is no motivation for creating two clauses in such a situation, and thus the initiation of only one such clause can be motivated. But if only one clause is initiated before NP1 is attached, then NP1 must be attached to this clause, and this clause is necessarily subjacent to the clause which Dominates the NP to which it is bound. Thus, the grammar interpreter will behave as if it enforces the Subjacency Constraint.

As a concluding point, it is worthy of note that while the grammar interpreter appears to behave exactly as if it were constrained by the Subjacency principle, it is in fact constrained by a version of the Clausemate Constraint! (The Clausemate Constraint, long tacitly assumed by linguists but first explicitly stated, I believe, by Postal [Postal 64], states that a transformation can only involve constituents that are Dominated by the same cyclic node. This constraint is at the heart of Postal's attack on the constraints that are discussed above and his argument for a "raising" analysis.) The grammar interpreter, as was stated above, limits grammar rules from examining any node in the active node stack higher than the current cyclic node, which is to say that it can only examine clausemates. The trick is that a trace is created and bound while it is a "clausemate" of the NP to which it is bound in that the current cyclic node at that time is the node to which that NP is attached. The trace is then dropped into the buffer and another S node is created, thereby destroying the clausemate relationship. The trace is then attached to this new S node. Thus, in a sense, the trace *is* lowered from one clause to another. The crucial point is that while this lowering goes on as a result of the operation of the grammar interpreter, it is only implicitly lowered in that 1) the trace was never *attached* to the higher S and 2) it is *not* dropped into the buffer because of any realization that it must be "lowered"; in fact it may end up attached as a clausemate of the NP to which it is bound - as the passive examples presented earlier make clear. The trace is simply dropped into the buffer because its grammatical function is not clear, and the creation of the second S follows from other independently motivated grammatical processes. From the point of view of this processing theory, we can have our cake and eat it too; to the extent that it makes sense to map results from the realm of processing into the realm of competence, in a sense *both* the clausemate/"raising" and the Subjacency positions are correct.

Evidence for the Determinism Hypothesis

In closing, I would like to show that the properties of the grammar interpreter crucial to capturing the behavior of Chomsky's constraints were originally motivated by the Determinism Hypothesis, and thus, to some extent, the Determinism Hypothesis explains Chomsky's constraints.

The strongest form of such an argument, of course, would be to show that (a) either (i) the grammar interpreter accounts for *all* of Chomsky's constraints in a manner which is conclusively universal or (ii) the constraints that it will not account for are wrong and that (b) the properties of the grammar interpreter which were crucial for this proof were *forced* by the Determinism Hypothesis. If such an argument could be made, it would show that the Determinism Hypothesis provides a natural processing account of the linguistic data characterized by Chomsky's constraints, giving strong confirmation to the Determinism Hypothesis.

I have shown none of the above, and thus my claims must be proportionately more modest. I have argued only that important sub-cases of Chomsky's constraints follow from the grammar interpreter, and while I can show that the Determinism Hypothesis strongly *motivates* the mechanisms from which these arguments follow, I cannot show necessity. The extent to which this argument provides evidence for the Determinism Hypothesis must thus be left to the reader; no objective measure exists for such matters.

The ability to drop a trace into the buffer is at the heart of the arguments presented here for Subjacency and the SSC as consequences of the functioning of the grammar interpreter; this is the central operation upon which the above arguments are based. But the buffer itself, and the fact that a constituent can be dropped into the buffer if its grammatical function is uncertain, are directly motivated by the Determinism Hypothesis. Given this, it is fair to claim that if Chomsky's constraints follow from the operation of the grammar interpreter, then they are strongly linked to the Determinism Hypothesis. If Chomsky's constraints are in fact true, then the arguments presented in this paper provide solid evidence in support of the Determinism Hypothesis.

Acknowledgments

This paper summarizes one result presented in my Ph.D. thesis; I would like to express my gratitude to the many people who contributed to the technical content of that work: Jon Allen, my thesis advisor, to whom I owe a special debt of thanks, Ira Goldstein, Seymour Papert, Bill

Martin, Bob Moore, Chuck Rieger, Mike Genesereth, Gerry Sussman, Mike Brady, Craig Thiersch, Beth Levin, Candy Bullwinkle, Kurt VanLehn, Dave McDonald, and Chuck Rich.

This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research Contract N00014-75-C-0643.

BIBLIOGRAPHY

Akmajian, A. and F. Heny [1975] *An Introduction to the Principles of Transformational Syntax*, MIT Press, Cambridge, Mass.

Bresnan, J. W. [1976] "Evidence for a Theory of Unbounded Transformations", *Linguistic Analysis* 2:353.

Chomsky, N. [1973] "Conditions on Transformations", in S. Anderson and P. Kiparsky, eds., *A Festschrift for Morris Halle*, Holt, Rinehart and Winston, N.Y.

Chomsky, N. [1975] *Reflections on Language*, Pantheon, N.Y.

Chomsky, N. [1976] "Conditions on Rules of Grammar", *Linguistic Analysis* 2:303.

Chomsky, N. [1977] "On Wh-Movement", in A. Akmajian, P. Culicover, and T. Wasow, eds., *Formal Syntax*, Academic Press, N.Y.

Fillmore, C. J. [1968] "The Case for Case" in *Universals in Linguistic Theory*, E. Bach and R. T. Harms, eds., Holt, Rinehart, and Winston, N.Y.

Gruber, J. S. [1965] *Studies in Lexical Relations*, unpublished Ph.D. thesis, MIT.

Jackendoff, R. S. [1972] *Semantic Interpretation in Generative Grammar*, MIT Press, Cambridge, Mass.

Marcus, M. P. [1977] *A Theory of Syntactic Recognition for Natural Language*, unpublished Ph.D. thesis, MIT.

Marcus, M. P. [1978] "Capturing Linguistic Generalizations in a Parser for English", in the proceedings of *The 2nd National Conference of the Canadian Society for Computational Studies of Intelligence*, Toronto, Canada.

Newell, A. and H.A. Simon [1972] *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, N.J.

Postal, P. M. [1974] *On Raising*, MIT Press, Cambridge, Mass.

Pratt, V. R. [1973] "Top-Down Operator Precedence", in the proceedings of *The SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, Boston, Mass.

Sager, N. [1973] "The String Parser for Scientific Literature", in [Rustin 73].

Winograd, T. [1971] *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*, Project MAC-TR 84, MIT, Cambridge, Mass.

Woods, W. A. [1970] "Transition Network Grammars for Natural Language Analysis", *Communications of the ACM* 13:591.