North East Linguistics Society

Volume 15
Issue 1 *NELS 15*

Article 3

1985

Deterministic Parsing: A Modern View

Robert C. Berwick
Massachusetts Institute of Technology

Amy S. Weinberg
University of Maryland, College Park

DETERMINISTIC PARSING: A MODERN VIEW

ROBERT C. BERWICK
Artificial Intelligence Lab, Massachusetts Institute of Technology
AMY S. WEINBERG
Department of Linguistics, University of Maryland, College Park

In the past five years there has been a dramatic conceptual shift in character of transformational grammars. No longer do grammars consist of extensive rewrite rules, tailored to particular languages. This leaves open the question of what parsers for these newer theories look like. In recent work, summarized in *the Grammatical Basis of Linguistic Performance*, we argued that one can gain considerable insight into parser design and the functional explanation of linguistic constraints if one assumes a direct match between grammatical principles and parser. We showed that *if* one is willing to make a few strong but natural assumptions about constraints on human parsing abilities and how grammars are used as parsers, then one can show, in part, why locality constraints like Subjacency must be a part of grammatical descriptions. Our assumptions were these:

- Parsing is *deterministic*, in the sense that once information about the structure of a sentence is written down, it is never retracted. This means that the information about a sentence is monotonically preserved as we analyze it.
- Grammatical representations are embedded *directly* into parsers, without intervening derived predicates or multiplied-out rule systems. This is an assumption of *transparency*.
- The human brain is finite.

In this paper we refine both parts of this argument. In the first half of the paper, we will show that grammatical theory can help parser design. We will show that if one embeds $\overline{X}$ theory directly into a parser, one gets a better parser. In the second part of the paper we turn to the functional explanation of locality principles. Here we will show that a deterministic parser can explain some of the properties of grammatical constructions. In particular, we show that our parsing model explains why parasitic gaps must be licensed at S-structure, and governed by Subjacency.

1. DETERMINISTIC PARSING and GRAMMAR

To begin, we discuss the new parser design. We use roughly the same bounded-context design outlined by Marcus (1980) and modified in Berwick and Weinberg (1984). At any given point in the analysis of a sentence, the parser has access to certain bounded left- and right-context information; this is used to fix the parser's next move. The parser is deterministic, in a sense made precise below. The left-context represents the analysis of the sentence seen so far, while the right-context holds words. In the Berwick and Weinberg model, left-context could consist of at most a single constituent domain plus at least one addition cyclic node domain; within these constituent domains, only nodes c-commanding the current token scanned are accessible. Thus the left-context is literally bounded in this model. We shall adopt these restrictions here. In the Marcus design, and in Berwick and Weinberg (1984), the right-context was taken to be at least the current word being analyzed plus (usually) two lookahead symbols. In the new model, there will be no lookahead; only the current token scanned is visible.

The new parser will be deterministic, in the following sense. At any step i in a parse we denote the representation of the sentence analysis at that point by A_i. A parser will be said to be *deterministic* if and only if the sequence of sets $A_1, A_2, \ldots$, is monotonically increasing in an information theoretic sense: each set in the sequence is either the same as or is a refinement of its predecessor. This means that one cannot *remove* information from an analysis set once it is placed there; one may, of course, refine this information. We shall see that this constraint has important implications for what grammatical operations can and cannot be represented in the new parser design.

So far, aside from the lack of lookahead, the parser design we have sketched is identical to that in Berwick and Weinberg (1984). We now turn to the key difference between the two models. Instead of reconstructing explicit tree structure as a part of its left-context representation and as its final representation of sentences, the new parser will build as its output representation a *phrase marker* in the original sense used in transformational grammar: a set representing all and only the *is-a* and *left-to-right precedence* relations in a sentence. This representation is augmented to directly encode the key $\overline{X}$ constraint identifying heads of phrases with lexical heads and the indexing of, e.g., traces.

An example that will be used later on will help to pin this notation down. Suppose we have the sentence, *I know that block supports the pyramid*. First, we shall number

the positions between the word tokens:

0 I 1 know 2 that 3 block 4 supports 5 the 6 pyramid 7 • 8

Here is the output set (using traditional *is-a* symbols for now) that we want. Each element of the set is a triple, consisting of first the phrasal name, followed followed by its *start* and *stop* positions. (Nothing crucial hinges on the exact details of this phrasal analysis.)

{ (S 0 7) (NP 0 1) (PRONOUN 0 1) (INFL 1 1)
(V 1 2) (VP 1 7) (DET 2 3) (NP 2 4) (NBAR1 3 4) (NOUN1 3 4)
(S 2 7) (INFL 4 4) (V 4 5) (VP 4 7) (NP 5 7) (DET 5 6)
(NBAR 6 7) (NOUN 6 7) }

Note that this information will allow one to reconstruct hierarchical relationships among the elements, if desired, though remaining ambiguous in the case of non-branching domination, since then we would have, e.g., (NP 1 3). (NP 1 3). (Indeed, in the output representation these two NPs would be merged and thus be quite literally indistinguishable.)

This representation, in fact, is exactly that used by so-called "tabular" parsing methods, such as Earley's algorithm. At every step in a parse, we simply *assert* which phrasal analyses are compatible with the input seen so far. Note that at any stage the analysis is simply a *set* of assertions comprising *is-a* and left-to-right precedence relationships compatible with the input seen so far. In fact, this is entirely equivalent to the *prefixes* of phrase markers, in the traditional sense (only prefixes, of course, since the parser cannot tell us anything about input that it has not seen so far). By the end of the parse, the assertion set will simply be the phrase markers compatible with the input string.

Information recorded at every step in the parse will be monotonically preserved. Once a recorded decision about the *is-a* or start and end points of an element can no longer be refined, this commitment cannot be revoked. That is, once we have fixed that an element of the assertion set is an NP, that cannot be changed to VP. Similarly, a decision that the NP extends between positions 1 and 3 of the input cannot be retracted once it is made. However, it is possible to *refine* a decision in an information theoretic sense, in two ways: first, one can add features to an is-a element, using the $\overline{X}$ features (e.g., changing an XP, with no features, to a +N item, to a +N −V item), provided that the element is still part of the left-context; second, we introduce a special asterisk notation, *, indicating that the *end* of a constituent has not yet been seen (this is apparently inevitable in a left-to-right analysis); third, we may *add* a new assertion, provided that the relevant left-context is still visible. The asterisk, since it represents *no* information about the end position of a constituent, may be refined by replacing it with a specific numerical value, but of course, once this is done, the numerical value may not be retracted.

Unlike the Marcus machine, the parser will not use lookahead. Instead, its parsing decisions are based entirely on the current word token scanned and the representation

of the input built up so far (the so-called left context of the parse). We may restrict the left-context, as is done in Berwick and Weinberg, to a literal encoding of what amounts to one S, but this won't be necessary in the sequel here. At each step in the parse, the assertion set of triples maintained will simply be the intersection of all triples compatible with the input seen so far. Thus at each step, the assertion set will encode just what is *known for certain* given the input so far processed. (This is a precise way to formulate the "wait and see" approach suggested by Marcus.)

Finally, the parser will be able to make top-down predictions (as in Marcus's parser) based on, e.g., subcategorization information, as when the lexical entry for a verb of a certain type sparks an expectation that a certain argument structure is expected. And, as in Marcus's design, the parser can certainly create assertions based on the actual words it finds in the input.

The best way to see how this works is to follow an example, comparing the operation of the new machine to the Marcus design. We'll do a very simple example first, just to get the idea down, and then show how $\overline{X}$ theory helps.

Our simple example uses conventional category names and a rather conventional context-free grammar. We now parse the sentence*I know that block supports the pyramid*. At step 0, before any input is seen, the parser can make the following different assertions (all output from here on is from an actual working version of the parser, written by Ed Barton). The control mechanism is simply that of a standard Earley-type parser. Nothing hinges on the details of this grammar, which is meant just to illustrate the operation of the machine; it is a typical context-free grammar for these sorts of phrases. "PL" stands for plural, and "SG" for singular.

(Active structure assertions in state set 0, after input () consumed.)
There are 11 assertion sets:

1. ((SBAR 0 *) (S 0 *) ((NP PL) 0 *) (ADJ 0 *))
2. ((SBAR 0 *) (S 0 *) ((NP PL) 0 *) ((PL DET) 0 *))
3. ((SBAR 0 *) (S 0 *) ((NP PL) 0 *))
4. ((SBAR 0 *) ((NP SG) 0 *) (S 0 *) ((NP PL) 0 *))
5. ((SBAR 0 *) ((NP SG) 0 *) (S 0 *) ((NP PL) 0 *) ((PL DET) 0 *))
6. ((SBAR 0 *) ((NP SG) 0 *) (S 0 *) (ADJ 0 *))
7. ((SBAR 0 *) ((NP SG) 0 *) (S 0 *) ((NP PL) 0 *) (ADJ 0 *))
8. ((SBAR 0 *) ((NP SG) 0 *) (S 0 *) ((SG DET) 0 *))
9. ((SBAR 0 *) ((NP SG) 0 *) (S 0 *) ((SG PRONOUN) 0 *))
10. ((COMP 0 *) (SBAR 0 *) ((NP SG) 0 *) (S 0 *))
11. ((COMP 0 *) (SBAR 0 *))

These possibilities correspond simply to all the possible initial expansions, given no input has been seen. (E.g., we could have an NP starting an S, with or without a determiner; a pronoun; etc. We have excluded many other possibilities for the sake of illustration.)

Now comes the key step: we intersect all these potential assertion sets to come up with what the parser can say it knows for certain. Here, this yields just one element, which becomes the first assertion set:

Here's the grammar for the example.

```
SBAR ==>  COMP S
      |  S
S ==>  (NP SG) (SG VP)
      |  (NP PL) (PL VP)
(NP SG) ==>  (SG PRONOUN)
      |  SBAR
      |  (SG DET) ADJ (SG NBAR)
      |  (SG DET) (SG NBAR)
      |  ADJ (SG NBAR)
      |  (SG DET) (SG NBAR)
(NP PL) ==>  (PL PRONOUN)
      |  (PL DET) ADJ (PL NBAR)
      |  (PL DET) (PL NBAR)
      |  ADJ (PL NBAR)
      |  (PL DET) (PL NBAR)
(SG NBAR) ==>  (SG NOUN)
(PL NBAR) ==>  (PL NOUN)
(SG DET) ==>  THE
      |  THAT
(PL DET) ==>  THE
      |  THOSE
COMP ==>  THAT
ADJ ==>  GREEN
      |  BIG
      |  RED
(SG NOUN) ==>  BLOCK
      |  PYRAMID
      |  SHEEP
(PL NOUN) ==>  BLOCKS
      |  PYRAMIDS
      |  SHEEP
(SG PRONOUN) ==>  I
      |  ME
(SG VP) ==>  (SG V) NP
(PL VP) ==>  (PL V) NP
(SG V) ==>  SUPPORTS
      |  KNOW
      |  SURPRISES
(PL V) ==>  SUPPORT
```

((SBAR 0 *))

Now we read *I*.

(Active structure assertions in state set 1, after input (I) consumed.)

There are 2 assertion sets:

1. (((SG V) 1 *) ((SG VP) 1 *) (S 0 *)
((NP SG) 0 1) ((SG PRONOUN) 0 1) (SBAR 0 *)
((NP SG) 0 *))
2. (((SG V) 1 *) ((SG VP) 1 *) (S 0 *)
((NP SG) 0 1) ((SG PRONOUN) 0 1) (SBAR 0 *))

Assertion set #2 is a subset of #1. The 2 sets have the following assertions in common:

(((SG V) 1 *) ((SG VP) 1 *) (S 0 *)
((NP SG) 0 1) ((SG PRONOUN) 0 1) (SBAR 0 *))

The common assertions coincide with assertion set 2.

After reading *know*, we can make the following assertions. Note here that NP is a proxy for the verb complement of *know*; we shall see how to fix this in a minute.

(((SG VP) 1 *) ((SG V) 1 2) (S 0 *)
((NP SG) 0 1) ((SG PRONOUN) 0 1) (SBAR 0 *)
(NP 2 *))

The next word is *that*. Here, the most that can be asserted for certain is the following. Note that *that* cannot be fixed as any sort of element as yet.

(((NP SG) 2 *) ((SG VP) 1 *) ((SG V) 1 2) (S 0 *)
((NP SG) 0 1) ((SG PRONOUN) 0 1) (SBAR 0 *))

Now we'll cover the rest of the parse in rapid-fire fashion, just to show the output. After *block*:

(((NP SG) 2 4) ((SG DET) 2 3) ((SG NBAR) 3 4)
((SG NOUN) 3 4) ((SG V) 1 2) ((NP SG) 0 1)
((SG PRONOUN) 0 1))

After *supports*:

```
(((SG VP) 4 *)      ((SG V) 4 5)          (S 0 *)
((NP SG) 0 1)       ((SG PRONOUN) 0 1)    ((SG V) 1 2)
((NP SG) 2 4)       ((SG DET) 2 3)        ((SG NBAR) 3 4)
((SG NOUN) 3 4)     (SBAR 0 *)            (NP 5 *))
```

the:

```
(((SG VP) 4 *)      ((SG V) 4 5)          (S 0 *)
((NP SG) 0 1)       ((SG PRONOUN) 0 1)    ((SG V) 1 2)
((NP SG) 2 4)       ((SG DET) 2 3)        ((SG NBAR) 3 4)
((SG NOUN) 3 4)     (SBAR 0 *)            (DET 5 6)
(NP 5 *))
```

pyramid:

```
((SBAR 0 7)         (S 0 7)               ((NP SG) 0 1)
((SG PRONOUN) 0 1)  ((SG V) 1 2)          ((NP SG) 2 4)
((SG VP) 4 7)       ((SG DET) 2 3)        ((SG NBAR) 3 4)
((SG NOUN) 3 4)     ((SG V) 4 5)          ((NP SG) 5 7)
((SG DET) 5 6)      ((SG NBAR) 6 7)       ((SG NOUN) 6 7))
```

So far, this may seem mundane and uninspiring. Note, though, how the system does not make any decisions it needs to retract. This will be true even if the sentence is ambiguous. So for instance, in such well-known examples as PP attachment ambiguity, since several analyses are equally possible, none will be asserted and the sentence will be left with its minimal phrasal analysis, without deciding at all that the PPs ought to be attached "high" or "low." What is good about this modular approach is that since the assertion sets are simply that—sets—we are free to filter them via any predicate whatsoever, in order to fold in the effects of lexical preferences, or whatever one wants.

Now consider some examples that Marcus claimed demanded lookahead. Note that our current design doesn't use lookahead at all—just the current token of the input (Marcus erroneously calls this a single cell of lookahead). We shall show that the $\overline{X}$ theory comes to the rescue here.

In English, words such as *for* and *that* are ambiguous. *For* can be either a preposition or a complementizer; *that* can be a pronoun, determiner, or complementizer. What must happen for a parser to successfully distinguish between these various uses?

I know that block supports a pyramid. (*that* is a determiner)

I know that blocks support the pyramid. (*that* is a Complementizer)

Examples such as these led Marcus (1980) to argue that one cannot parse deterministically without looking at one or more tokens ahead into the input, to see whether *blocks* is singular or plural. If we cannot use lookahead, and we must decide whether to parse *that* as part of a Noun Phrase or a new Sentence, we are stuck. This is also true of our explicit rewrite rule approach, as used in the new parser design. If

the only nonterminals names are objects such as Complementizer and NP, and if we must choose between these without retraction, then we are in trouble. (This isn't to say that Marcus wasn't in trouble here. His parser does work on these examples, but rather bizarrely. Indeed, the "diagnostic" rule to handle *that* is among the most baroque in his entire grammar: it specifically violates the general left-to-right operating principle of his machine, attaching the *first* item in the input buffer to the ***second***.

Suppose, though, the the $\overline{X}$ representational format is directly incorporated in the parser. Then parsing can proceed deterministically, or, rather, monotonically. In particular, suppose that we adopt the idea that a verb such as *know* subcategorizes for some XP, and that all Heads project to form phrases of some kind. Now let us see what happens when the parser analyzes a sentence such as, *I know that the block supports a pyramid*. We pick up the action at the time just after *know* is scanned. Since *know* predicts that an XP starts after it, we add the assertion triple (X-max 2, 8) to the assertion set.

Now what? Reading *that*, the parser's compatible assertions include: (*that*, with no features, or perhaps +N, 2, 3); (X-max, 2, *) (projected from *that*, now assuming the $\overline{X}$ theory); (X-max, 2, *); and the remainder from the earlier portions of the parse. Note that the features of the X-max, and *that* remain underdetermined; this is the best the machine can do at this point.

The parser now scans *block*. We add the triple (*block* +N −V, 3, 4); but we can now also refine the earlier triples, (X-max, 2, *) to (X-max +N −V , 2, *). Finally, by scanning *support* we can then add whatever refinement we want to the first X-max triple, turning it into an S-bar (or S, as one might wish in some cases). Eventually, the X-max (NP) will span positions 2 through 4; the S-type phrase, 2 through 7.

Now let us try *I know that blocks support the pyramid*. Again we predict a (X-max 2, *) triple after *know*, and (X-max, 2, *); (*that*, 2, 3). Now the parser sees a plural Noun, *blocks*. Based on this information, it can now refine the X-max triple to (X-max + whatever feature complex S-bars have, 2, *); (*that* + features for COMP, 2, 3). We also add the new projection (X-max +N −V, 3, *) (for the NP headed by *blocks*), and one for an S; note that this is always allowed. Eventually, the S-bar will stretch from position 2 through 7; the S, from position 3 through 7.

Note that no lookahead is required to resolve these cases, nor any abnormal rule that attaches items in a peculiar fashion. The "wait and see" or "attention shifting" behavior of the Marcus parser does not need to be stipulated; rather, it falls out of the normal behavior of the parser, operating with $\overline{X}$ type rule features. Where the $\overline{X}$ system underdetermines feature complexes, we get the *effect* of wait-and-see behavior, but it is part and parcel of the normal operation of the machine—nothing special at all need be said in order to get this effect.

The same approach resolves *for* ambiguities. If the grammar has to use the categories PP, $\overline{S}$, and so forth, then our deterministic parser must fix this categorial status correctly, without extra lookahead in the proposed model. But then the parser cannot correctly decide between *for* as the Head of a Prepositional Phrase and *for* as the a Complementizer—the Head of an $\overline{S}$, in some theories. Now suppose that the $\overline{X}$

schema is directly used by the parser. Then, given that *for* is a Head of some kind, the parser can create a maximal projection, with a kind of "operator" status (the Head element). The features of this projection can remain partially determined until the following material is analyzed as either an NP followed by verbal material (then *for* heads an $\bar{S}$) or just an NP (*for* heads a PP). In fact, on this alternative, there is no "PP" or "$\bar{S}$" at all, just the Head with whatever features it has.

Let's take one more example and then stop. Consider another sentence type where a verb can subcategorize for either NP or Sbar, e.g, *I expect John/I expect John to leave*. Exactly the same approach will work here as before. In the first sentence, the X-max Recipient triple will eventually be co-identified with the N-max projection, via refinement. In the second, the X-max will be refined to be an S-bar spanning positions 2 to 5, while the N-max will stretch only from position 2 to position 3 in the input. Refinement occurs automatically as each token is scanned: when the infinitive *to* is reached, the N-max is closed off at position 3 and its N-max features not percolated to the X-max. No lookahead is required.

The framework sketched here is perfectly compatible with constraints that allow the order among complements to, e.g., a verb, to be free, if there is no other constraint on them. That is, in order to establish what constituents are to be expected after a verb, we simply project, directly, the lexical subcategorization frame of the verb; there is no need for an ordered right hand side as in a traditional context-free rule. Whatever assertions are compatible with the (unordered) subcategorization frame will simply be made as before. Of course, if there *are* constraints, such as case assignment/adjacency considerations, these may be enforced and the resulting assertion sets filtered; no change in the basic parsing machinery is required. Also, because no lookahead is ever used, "short" garden paths, such as *the boat floated sank* are at least possible; the Marcus parser had to deliberately ignore its own lookahead information in an ad hoc way in order to garden path in such cases.

The new parsing design takes $\overline{X}$ theory seriously. By doing so, it can avoid problems that plague parsers built on explicit rule systems. It exhibits the right kind of "wait and see" behavior, while remaining true to the principle of determinism.

2. Parsing and grammatical constraints

In the previous section we showed how the problem of encoding grammatically relevant information can help us constrain the design of the parsing algorithm. We can approach the problem from the opposite perspective. We can ask what conditions efficiency imposes on grammatical theories. In this section we will show that two otherwise mysterious properties of the parasitic gap construction receive a functional explanation—if we assume that the grammar is embedded with no additional ad hoc mechanisms into a deterministic parser.[1] In particular, we can explain why parasitic gaps *must* be licensed at S-structure. In addition, we explain why the parasitic gap construction is governed by *subjacency*.

In Berwick and Weinberg (1984) we discussed a subclass of parsers known as Bounded Context parsers. The three most relevant properties of these parsers are:

1. They are extremely efficient. They can parse sentences in linear time.

2. This efficiency is guaranteed in part by their deterministic operation.

3. They use only a literal representation of the *left context* of a parse, that is, a representation of the structure seen so far that does not use essential variables.[2]

We argued that if we could show that natural language could be parsed by such a machine, then it could be parsed efficiently. We then examined problems that natural language presents to such a device.

As described in the previous section, cases of structural ambiguity pose the greatest challenge to a deterministic parser. Consider a sentence like (1) in this light.

1. What$_i$ do you think Fred ate e$_i$.

A correct analysis of this sentence entails that the parser inserts an empty category after the verb. Note however that this verb has an ambiguous subcategorization and takes an intransitive reading as well.

2. Did you think Fred ate?

A deterministic answer to the question of whether to insert an empty category after the verb cannot be obtained simply by the analysis of the local subcategorization environment. Rather, the only firm evidence for the expansion of an empty category in (1) is the presence of the *wh* operator at the beginning of the sentence. But this disambiguating element can be arbitrarily far from the empty category, as examples like (3) show.

3. What$_i$ do you believe Fred said that Bill believed that John expected Sue to eat e$_i$?

4. Do you believe that Fred said that Bill believed that John expected Sue to eat?

This suggests that if the parser had to consult the *wh* element at the beginning of the sentence, it might have to store a potentially unbounded left context. This is because we cannot guarantee that a finite number of symbols intervene between the trace and antecedent. (We can't encode the left context using essential variables.) This entails some kind of bound on the distance between the empty category and the element that binds it.[3]. This gives us the effect of cyclic movement and subjacency. Assuming the parsing analog of successive cyclic movement, the decision to insert a trace in a structure like (3) can be made by looking at a finite amount of left context. If there is a trace in the subjacent COMP we can expand the phrase structure with a trace; if not, we don't.[4].

5. What$_i$ do you believe [e$_i$ [John said ...[e$_i$ [Bill would eat ...]]]]

In addition we made the assumption that the parser shunted nodes from the left context as they were completed. For example, as a verb phrase is completed its daughters are removed from the left context. In Berwick and Weinberg (1984) we showed that this gives the effect that only c-commanding elements can serve as the left context for phrase structure expansion. In our book we presented some psycholinguistic evidence for this assumption and also showed that it could be used to correctly delimit why binding relationships are governed by c-command.

As pointed out by Janet Fodor (forthcoming) and contra Berwick and Weinberg (1984), parasitic gaps present our parser design with the same ambiguities that motivated subjacency for regular *wh* movement. That is, there are structures where the decision to create a parasitic gap or not must depend on the presence or absence of certain left context material. This is illustrated in (6) and (7):

6. This is the meal$_i$ that I cook without eating e$_i$.

7. Can you cook a meal without eating?

Chomsky (contra Fodor (forthcoming) and Berwick and Weinberg (1984)) provides the following examples demonstrating that in fact parasitic gap constructions are governed by Subjacency. Examples (8), (9), and (10) parallel overt movement in that movement from complex noun phrases is unacceptable.

8. Who–i did you read a book about e$_i$ to e$_i$?

9. Which man$_i$ did you interview e$_i$ without reading up on e$_i$?

*10. Which man$_i$ did you interview e$_i$ without reading [$_{NP}$ [the file]$_j$ [$_S$ you made e$_j$ on e$_i$]]?

In (8), both gaps are subjacent both from the complementizer, and from each other. This is shown by both (11) and (12), where overt movement from both the parasitic and regular gap positions is acceptable.

11. Who$_i$ did you read a book about e_i.

12. Who$_i$ did you read the book (that Mary bought yesterday) to e_i.

Chomsky uses the contrast in (9) and (10) to argue that parasitic gaps are bound to empty operators and are licit only if they are subjacent to these operators. These empty operators are interpreted as marks of predication and so must appear at the head of the adjunct clause. Put in terms of our parsing model, we can use the presence of the overt operator to signal the presence of the "real" gap. The placement of the empty operator is governed by the independent principles of either predication or A′ binding. The presence of the empty operator, in turn, can be used to signal the presence of the parasitic gap, if it is in a subjacent position. In addition, Chomsky assumes that every ungoverned category counts as a bounding node.[5] This analysis predicts that (10) is bad because, as a sign of predication between the relative clause

and the head of the complex NP, the empty operator inside this relative must be bound to (coindexed with) the head. Coindexing the parasitic gap to this operator as well will result in an ill-formed structure, because quantifiers cannot be bound to two variables, (13). Neither the overt operator at the head of the sentence, nor the empty operator at the head of the adjunct are subjacent to the gap, and so they cannot license it. Therefore this structure is ruled out. This contrasts with (9), where every trace is subjacent to the operator that licenses it, as shown in (14).

*13. Which man$_i$ [$_S$ did you [$_{VP_j}$ interview e_i][$_{PP}$ without [OP2$_j$ [$_S$ PRO reading [$_{NP}$ the file$_i$ [$_{S'}$ OP$_i$ [$_S$ that you made on$_j$ e_i]]]]]]]

14. Who$_i$ [$_S$ did you [$_{VP}$ interview e_i][$_{PP}$ without [$_{S'}$ OP$_i$ [PRO reading up on e_i]]]]]?

Assuming that we can show that the creation of empty operators causes no problems for a deterministic system, we can use their presence to license parasitic gaps in the appropriate structures. Thus we can make the parsing model predict the properties of this construction in a straightforward and independently motivated way. It is important to note at this point that we are not changing assumptions in an ad hoc way simply to model the facts. The problem with our first attempt was that we did not follow the logic of our predictions clearly. The model *actually* predicts that parasitic gaps *should* be governed by Subjacency, as Fodor notes in her forthcoming article.

We now present an algorithm to create empty operators that is also compatible with a deterministic approach. Note that the case of empty operators in adjuncts is similar to the case of factive Noun Phrases cited by Fodor in her criticism of Marcus. As in factives, the presence of the overt operator makes parasitic gaps *possible* in adjunct positions, but it does not make them obligatory in these structures. Consider (15)–(17).

15. Who did you meet without greeting.

16. Who did you meet without greeting him.

17. Who did you meet without clearing the rendezvous with security.

In a case like (15), the parser must place an empty operator in the complementizer of the adjunct phrase in order to bind the empty parasitic object of the verb *greeting*. In (16) and (17) by contrast, we do not want to place an empty operator in this position, because there is no parasitic gap in the adjunct for the operator to bind.[6] In (16) the parasitic gap is filled by a pronoun and in (17), there is no corresponding gap position at all. Because of the possibility of successive cyclic movement however, the parasitic gap can be indefinitely far away on the surface from the empty operator position. A deterministic parser with limited lookahead will not be able to wait for the disambiguating right context. Therefore, there will be certain cases where it will incorrectly place an empty operator in the adjunct's COMP.

Fodor implies that these facts pose a problem solely for deterministic parser, suggesting that a nondeterministic solution is called for. In fact, the deterministic/nondeterministic issue is beside the point. If the distinction is between a deterministic parser and a nondeterministic parser that backtracks (Fodor's choice), then both will have problems because they both at least superficially predict that such cases cause people to have noticeable difficulties in comprehending these sorts of sentences. But none of (15)–(17) are difficult to understand.

Nondeterministic parsers with backtracking partially equate the difficulty in comprehending a sentence with the amount of backtracking needed from the point of disambiguation to the point where a parsing error was made.[7] Thus, *John was hit by Mary* is easy to comprehend even under the assumption that the parser would first hypothesize that this is an active sentence, because the parser recognizes its mistaken characterization of *John* as an Agent as soon as it sees the passive predicate. In contrast, *Who did you ask to sing that stupid song for* is more difficult, because there is a tendency to interpret *who* as the Subject of *ask*. The disambiguation point is quite far from this decision point, so when the parser realizes its mistake it must unwind through a long stretch of material to make the appropriate corrections.[8]

Cases like (15)–(17) cause problems for the backtracking approach because they break the association between the extent of backtracking necessary to correct false starts and perceived sentence complexity. None of the examples in (15)–(17) produce processing complexity. This shows that there is not even a preference for adjuncts with or without parasitic gaps. Whatever the first hypothesis of the (deterministic or backtracking) parser—whether it inserts an empty operator in the adjunct's complementizer or not—one of the structures is incorrectly predicted to be difficult to process because of extensive backtracking from the site of the disambiguating parasitic gap or end of the adjunct needed to correct the mistake. (18a) and (18b) show that no extra processing complexity is observed even in cases where the disambiguating right context is very far away from the point where the decision about whether to insert an empty operator must be made.

18a. Who did you search for without telling Sue to convince Bill to ask Harry to come with you?

18b. Who did you search for without telling Bill to ask Sue to inform Harry that you would meet?

It seems then that these kind of sentences are problems for both deterministic and nondeterministic (backtracking) parsers. We could solve them if we could design an algorithm in which the semantic component simply didn't interpret empty operators unless they were eventually bound to elements in argument positions. Since these elements have no phonetic content, if they received no semantic interpretation, it would be as if these elements never existed.[9] In that case we could insert the empty operator in all sentences, but we would be sure to be right because an unbound empty operator would simply be ignored, because it is invisible. In fact the two stage parsing model discussed in our book provides just such a mechanism.

We argued on conceptual and psycholinguistic grounds that the natural language processor was a two stage mechanism. The first stage dealt with tree expansion and the second dealt with indexation. In addition to having a different function, the second stage worked on a different representation. During the first stage, the completion of a category signaled the parser to shunt the category's daughter into a separate stack, which we called the Propositional Node Stack (PNS). The intuition behind this shunting was that once a category's thematic role was established from its position in the syntactic tree, the parser wouldn't need to retain many of the details of syntactic structure. We showed that elements in the same c-command domain are not put in the PNS until all categories in the domain are complete. This algorithm allowed the parser to correctly compute c-command relations between categories. This was crucial since these relations govern the application of the binding operations on the previously expanded tree. Pursuing the intuition that the PNS was a representation concerned with purely semantic aspects of the interpretation, we placed a semantic visibility condition on the categories appearing in this component. We claimed that to be interpreted by the semantic component (PNS), a category had to have semantic features. These were the features that allowed a Noun Phrase to either denote an individual or a set of individuals or allowed a quantifier to delimit a scope.[10] Assuming a category had such features it would be given a "referential index" and be visible in the PNS. If a category did not intrinsically have such features, it could obtain a referential index by being linked to an element that did.[11] Given the shunting procedure, an element would have to be in the same c-command domain as its antecedent in order to receive a referential index before being shunted into the PNS. If an element did not receive an index before shunting, it would become invisible and receive no interpretation. This allowed us to provide a principled explanation for the fact that grammatical conditions specifying *c-commanding* antecedents seem to apply *only* to categories with no independent referential status.[12] Chomsky (1981 and 1984 class lectures) has suggested that association with a thematic (theta) role is also a necessary condition on visibility for semantic interpretation roles. We will adopt Chomsky's suggestion and state the combined condition on visibility as follows.

> (Visibility Condition) To be visible in the PNS, an element must be associated with a theta role (either by occupying a theta position or binding an element in a theta position) and must have referential features (features that either designate an individual or set of individuals or that delimit a range).

We will now show that the independently motivated shunting procedure and visibility conditions give an account of empty operators that explains why they cause no processing difficulties.

Let us reconsider sentences (15)–(17). In (15), the parser recognizes that part of the sentence is an adjunct phrase. This signals the possibility of a parasitic gap in the subsequent structure. The parser therefore inserts an empty operator in the COMP position, as shown in (19):

19. Who$_i$ did you meet e$_i$ without [$_{S'}$ OP$_j$ ...

If the parser subsequently finds a gap position in a subjacent domain, it can create a trace and bind the operator to it, thus associating the operator with a theta position, as in (20).

20. Who$_i$ did you meet e$_i$ without [OP$_j$ [s greeting e$_j$]]

Before shunting into the propositional node stack, the operator must locate an antecedent in the c-command domain with a *referential* index. If it does not find one, then neither it nor its trace will be interpreted, because even though they are associated with a theta role, they are not associated with a category that delimits a range. In this case the overt operator *who* is present in the c-command domain, so both the empty operator and the trace can receive the category's referential index (i) and so be interpreted in the PNS.

Compare this to (21). In (21) below, the parser will also detect an adjunct and so create an empty operator. Assuming the verb in the adjunct can take an argument, the subjacent empty operator will license a parasitic gap.

21. Did you watch the movie without [s' OP$_j$ [s eating e$_j$]]

In this case there is *no* overt operator in the c-command domain (or anywhere else in the structure for that matter). Therefore, the empty operator and its trace are not associated with referential features and so will not be interpreted at PNS.

In cases like (17) above, the adjunct again triggers the creation of an empty operator. Since there is no gap in the adjunct phrase, the operator is not associated with a theta role. Therefore, even though there is an overt operator to link with, the empty operator does not meet the criterion for visibility at PNS and so is not interpreted.[13] Since empty operators are not interpreted unless both conditions on visibility are met, a deterministic parser can always create these categories because they can never force it to simulate nondeterminism either by backtracking or parallelism in order to correct for past mistakes. Note that this solution will only work for *empty* operators. Lexically specified elements will receive a phonetic interpretation but no semantic interpretation, a situation that will lead to unacceptability. An empty element with no semantic features, however, is neither semantically nor phonetically interpreted and so simply plays no role in the interpretation of the sentence.

The astute reader will have noted an apparent problem created by this solution. Why, one might ask, if empty categories can become invisible at later stages of interpretation, must we cue their creation to the presence of overt operators? The cases that motivated the account in the first place were those in which the local subcategorization of a verb was indeterminate. Before positing an empty element after such a verb, we claimed that we had to make sure that an actual operator was present in the previously analyzed structure. However, given our present approach, one might be tempted to argue that if a verb that can be optionally transitive turns out to be used intransitively in a given structure, the gap will simply not be associated with an operator and so become invisible in the PNS. This seems to dash the motivation for restrictions on left context, crucial for the functional motivation of Subjacency

in the first place. But it is only elements with no phonetic features that can escape unacceptability if they are not semantically interpreted. Since *wh* elements have case features,[14] they will be visible in the phonological component.[15] This makes certain predictions about the applicability of Subjaceny to NP movement. As noted in Lasnik and Saito (1984), all the cases where we seem to need Subjacency to rule out unacceptable NP movements are actually also ruled out *redundantly* by the Empty Category Principle. Under our approach, we *predict* that NP movement should not be governed by Subjacency, thus ruling out this redundancy, always a welcome result.[16]

Looking at the distribution of parasitic gaps from the parsing perspective allows us to supplement Chomsky's analysis in important ways. It allows us to *derive* the fact that parasitic gaps *must* be licensed at S-structure. That is, we derive as a theorem the fact that quantifiers and *wh* operators that move to COMP or some other pre-S position at LF do not create acceptable parasitic gap structures, as shown by examples (22) and (23).

*22. [S You [VP [VP met who$_i$] [PP without greeting e$_i$]]]

*23. [Everyone [VP [VP met someone$_i$][PP without greeting e$_i$]]]

We know independently that parasitic gap constructions are not licit if the real gap occurs in Subject position.[17] In addition, if our analysis is correct, the overt operator must occur in a c-commanding COMP. As mentioned, the c-command requirement is ensured by the shunting design of the parser. If an element does not c-command a category it is not visible to it and so cannot be used to create that category as we expand the parse tree. Neither the *wh* element, nor the quantifier in (22) or (23) c-commands the adjuncts containing the parasitic gaps. Given the above account, there will be no binder to give referential features to the empty operator in the COMPs of these adjuncts and thus neither they nor their traces will be interpreted in the PNS. Given that the input for parsing decisions is the S-structure of the sentence, the subsequent movement of a category to a c-commanding position at a post S-structure level cannot help the parser decide how to expand the parse tree. Our parsing theory can derive both the fact that Subjacency is an S-structure property and the Subjacent government of parasitic gaps along with their licensing at S-structure—the central properties of the construction.

Conclusions

We have seen that by linking grammatical theory and parsing theory closely together, we can help both. Parsing theory benefits from the constraints given by grammars: if it were not for locality principles and $\overline{X}$ theory, deterministic parsing would be difficult, if not impossible. On the other side, the pattern of locality constraints seems best explained as a fact about deterministic parsing, not a fact about the grammar. As expected, the more we learn about the representations underlying knowledge of language, the more we can know about how those representations are used, and why they take the form that they do.

FOOTNOTES

[1]This is a greatly condensed version of Weinberg (forthcoming).

[2]It is possible to obtain the effect of an essential variable by encoding left context generatively. This is the method used in GSPG. By adding complex nonterminal symbols (so-called "slashed categories") we can pass information that a potential filler has been encountered indefinitely far. This solution would be ad hoc for a parser implementing a GB style grammar. The only categories grammatically motivated are those independently generated by the $\bar{X}$ system (Head and their projections). The $\bar{X}$ system does not distinguish between projections of empty and lexical categories. In GB phrase structure configurations are purely theoretical constructs derived from independent subsystems, and there is no independent way to justify complex nonterminals. See Chomsky (1981) for justification. Since generative encoding is not available, we must ensure that there is a *literal* bound on the number of nonterminals that intervene between a trace and its antecedent.

[3]We cannot go into the argument here, but in Berwick and Weinberg (1984) we show that the noncounting property of grammars entails that this restriction applies to zero or one phrase domain, as opposed to say, six bounding nodes.

[4]This situation contrasts with lexical anaphors or LF movement. In structures like (a) or (b) below, the binding of the italicized element is indeterminate, but the decision about whether to create these categories is fixed without left context because they are phonetically realized. Therefore, we correctly predict that Subjacency doesn't apply here.

(a) Who do you think e ate ***what***

(b) The men liked ***each other***.

[5]Alternatively, following Aoun and Clark (1985), we can claim that empty operators count as A′ anaphors and so obey the locality conditions that apply to this class. See Weinberg (forthcoming) and Aoun, Hornstein, Lightfoot, and Weinberg (forthcoming) for details.

The analysis in the text contrasts with Chomsky (1982) where parasitic gaps are considered underlying PROs. Brody (1983) provides independent arguments showing this account of the distribution of parasitic gaps is inadequate because it relies on the so-called *functional definition* of empty categories. In addition, the earlier analysis would obviously not predict the observed distribution of the data, since PROs are typically not bound by operators, empty or otherwise.

[6]If these operators are available at all stages of comprehension then the fact that the empty operator has no variable to bind should make the sentence as bad as in (a):

(a) Who did John meet Mary?

[7]These are parsers like the standard ATN designs proposed by Woods (1970).

[8]See Frazier, Clifton, and Randall (1983) for details.

[9]An Alternative would obviously be to come up with an analysis that did not posit empty operators in these and related cases. Such an account is difficult to conceive of, because we would also have to account for the subjacency effects that these constructions exhibit. By this we do not mean coming up with an alternative

functional explanation for Subjacency in these cases. We mean allowing the parser (or the grammar) to distinguish those cases that are grammatical from those that do not obey the constraint.

[10]Examples of categories with intrinsic semantic features are proper names like *John*, pronouns like *him wh* phrases like *what* or *which man*.

[11]Categories that have no intrinsic semantic features and so can receive referential indices only by linking are bound anaphors like *each other* or *herself*, empty NP and *wh* traces, and certain non-*wh* quantified expressions. See Weinberg (forthcoming) for details.

[12]See Berwick and Weinberg (1984, pp. 173–182) for the conceptual arguments and Weinberg (forthcoming) and Weinberg and Garrett (forthcoming) for psycholinguistic results and additional consequences of this approach.

[13]This approach will also handle empty operators in *tough* movement, topicalization, relative clauses, and the factive NPs that Fodor discusses in her criticism of Marcus. As should be obvious, since all these structures also involve predication between a phrase and a head, topic, or adjective phrase, exactly the same logic applies. See Weinberg (forthcoming) for details.

[14]See Chomsky (1981) for justification of this assumption.

[15]See Aoun and Lightfoot (1984) for discussion.

[16]See Weinberg (forthcoming) for details. Note that the non-government of NP movement by Subjacency *reinforces* the point made in Berwick and Weinberg (1984)—namely, that Subjacency governs a natural class from the parsing perspective. The example just given shows that Subjacency only governs a *subset* of the movement constructions, the gapping examples discussed later on in this section show that Subjacency governs a *subset* of the deletion constructions. From a grammatical viewpoint, this is an entirely unnatural result.

This approach also makes sense of some preliminary results reported by Frazier (1984 Nels proceedings). Frazier claims that eye movement tasks suggest that subjects try to fill gaps using operators that are not subjacent to them, if the verbs governing the gap position are strongly subcategorized for direct objects. The cases are like those in (a):

a. *What$_i$ did [the girl [$_S$ who won e$_i$]] receive e$_i$

Given our approach we might claim that the gap inside the island is created on the basis of the empty operator in the COMP of the relative COMP. The fact that experimental subjects seem to look back to the overt *wh* element is compatible with our approach if we claim that this is the result of the attempt to bind this operator (an operation *not* governed by Subjacency) to the overt operator.

[17]See Chomsky (1982).

REFERENCES

Aoun, J. and R. Clark, 1985. On empty operators. University of Southern California Working Papers.

Aoun, J. and D. Lightfoot, 1984. Government and contraction. *Linguistic Inquiry*, 15, 465–473.

Aoun, J., N. Hornstein, D. Lightfoot, and A. Weinberg, forthcoming. Two notions of locality. University of Maryland College Park and University of Southern California, unpublished ms.

Berwick, R. and A. Weinberg, 1984. *The Grammatical Basis of Linguistic Performance*. Cambridge, MA: MIT Press.

Brody, M., 1983. On contextual definitions and the role of chains. *Linguistic Inquiry*, 15, 355–380.

Chomsky, N., 1981. *Lectures on Government and Binding*. Dordrecht: Foris Publications.

Chomsky, N., 1982. *Some Concepts and Consequences of the Theory of Government and Binding*, Cambridge, MA: MIT Press.

Fodor, J.D., 1985 forthcoming. Explanation and deterministic parsing. *Language and Cognitive Processes*, 1.

Frazier, L., Clifton, and J. Randall, 1983. Filling gaps: decision principles and structure in sentence processing. *Cognition*, 13: 187–222.

Frazier, L. 1985. NELS Proceedings, this volume.

Lasnik, H. and M. Saito, 1984. On the nature of proper government. *Linguistic Inquiry*, 15, pp. 235–290.

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

Weinberg, A. and M. Garrett, forthcoming. Sentence processing and C-command. Ms, Department of Linguistics, University of Maryland College Park, Department of Psychology, MIT.

Weinberg, A., forthcoming. Topics in the theory of linguistic performance, PhD dissertation, MIT Department of Linguistics and Philosophy.

Woods, W. 1970. Transition network grammars for natural language analysis. *Communications of the Association for Computing Machinery*, 13:591–606.