LEFT-CORNER PARSING AND PSYCHOLOGICAL PLAUSIBILITY

Philip Resnik
Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104, USA
resnik@linc.cis.upenn.edu

Abstract

It is well known that even extremely limited center-embedding causes people to have difficulty in comprehension, but that left- and right-branching constructions produce no such effect. If the difficulty in comprehension is taken to be a result of processing load, as is widely assumed, then measuring the processing load induced by a parsing strategy on these constructions may help determine its plausibility as a psychological model. On this basis, it has been argued [AJ91, JL83] that by identifying processing load with space utilization, we can rule out both top-down and bottom-up parsing as viable candidates for the human sentence processing mechanism, and that left-corner parsing represents a plausible alternative.

Examining their arguments in detail, we find difficulties with each presentation. In this paper we revise the argument and validate its central claim. In so doing, we discover that the key distinction between the parsing methods is not the form of prediction (top-down vs. bottom-up vs. left-corner), but rather the ability to instantiate the operation of composition.

1 Introduction

One of our most robust observations about language — dating back at least to the seminal work of Miller and Chomsky [MC63] — is that right- and left-branching constructions such as (1a) and (1b) seem to cause no particular difficulty in processing, but that multiply center-embedded constructions such as (1c) are difficult to understand.

1 a. [[[John's] brother's] cat] despises rats.
  b. This is [the dog that chased [the cat that bit [the rat that ate the cheese]]].
  c. #[The rat that [the cat that [the dog] chased] bit] ate the cheese.

The standard explanation for this distinction is a tight bound on space in the human sentence processing mechanism: center-embedded constructions require that the head noun phrase of each subject be stored until the processing of the embedded clause is complete and the corresponding verb is finally encountered.[1] Alternative accounts have been proposed, most sharing the premise that the parser's capacity for recursion is limited by bounds on storage. (See, for example, [Kim73] and [MI64]; for opposing views and other pointers to the literature see [DJK$^+$82].)

The distinction between center-embedding and left/right-branching has important implications for those who wish to construct psychologically plausible models of parsing. Johnson-Laird [JL83] observes that neither the top-down nor the bottom-up methods of constructing a parse tree fit the facts of (1), and proposes instead the less-well-known alternative of left-corner parsing. Abney and Johnson [AJ91] discuss a somewhat more general version of Johnson-Laird's argument, introducing the abstract notion of a parsing *strategy* in order to characterize what is meant by bottom-up, top-down, and left-corner parsing.

In this paper, we examine the argument as presented by Abney and Johnson and by Johnson-Laird, and point out a central problem with each variation. We then present the argument in a form that remedies those difficulties, and, in so doing, we identify a previously underrated aspect of the discussion that turns out to be of central importance. In particular, we show that the psychological plausibility argument hinges on the operation of *composition* and not left-corner prediction *per se*.

2 Comparing Strategies

2.1 Summary of the Argument

For expository purposes, we begin with the discussion in [AJ91]. Abney and Johnson assume, as we shall, that the human sentence processing mechanism constructs a parse tree, consisting of labelled nodes and arcs, incrementally over the course of interpreting an utterance, though the global parse tree need never "exist in its entirety at any point." They define a *parsing*

[1] This observation is by no means language specific, though in SOV languages it is embedding on objects, not subjects, that causes difficulty.

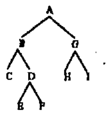
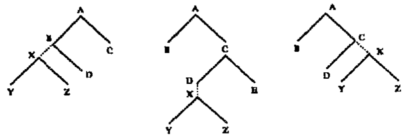


Figure 1: *A parse tree*



left-branching center-embedded right-branching

Figure 2: *Branching structures*

strategy to be "a way of enumerating the nodes and arcs of parse trees." This is, in fact, a generalization of the concept of a *traversal* [ASU86].

A *top-down* strategy is one in which each node is enumerated before any of its descendants are; a *bottom-up* strategy is one in which all the descendants of a node are enumerated before it is. So, for example, a top-down strategy would enumerate the nodes of the tree in Figure 1 in the order ABCDEFGHI, and a bottom-up strategy would enumerate them in the order CEFDBHIGA. In a *left-corner* strategy, for each node η, the leftmost child of η is enumerated before η, and the siblings of the leftmost child are enumerated after η. The strategy takes its name from the fact that the first item on the right-hand side of a context-free rule (its *left corner*) is used to predict the parent node. For example, having recognized constituent C in Figure 1, the parser predicts an invocation of rule $B \rightarrow C\,D$ and introduces node B. The complete left-corner enumeration of the tree is CBEDFAHGI.

Thus far, we have discussed only the order of enumeration of nodes, and not arcs. Abney and Johnson define as *arc-eager* any strategy that enumerates the arc between two nodes as soon as both nodes are present. An *arc-standard* strategy is one that enumerates the connecting arc once either none or all of the subtree dominated by the child has been enumerated. For example, the arc-eager left-corner enumeration of the tree in Figure 1 would introduce arc $\langle B, D\rangle$ just after node D was enumerated, while the arc-standard version of the left-corner strategy would first completely enumerate the subtree containing E, D, and F, and *then* enumerate arc $\langle B, D\rangle$.

In order to characterize the space requirements of a parsing strategy, two more definitions are required. A node is said to be *incomplete* either if its parent has not yet been enumerated (in which case the parser must store it until it can be attached to the parent node), or if some child has not yet been enumerated (in which case the parser must store the node until its child can be attached). The *space requirement* of a parsing strategy, given a grammar, is the maximum number of incomplete nodes at any point during the enumeration of any parse tree of the grammar.

Having established this set of definitions, the goal is to decide which parsing strategies are psychologically plausible, given the facts about the human parsing mechanism as exemplified by (1). The central claim is summarized in the following table:

Strategy		Space required		
Nodes	*Arcs*	*Left*	*Center*	*Right*
Top-down	either	$O(n)$	$O(n)$	$O(1)$
Bottom-up	either	$O(1)$	$O(n)$	$O(n)$
Left-corner	standard	$O(1)$	$O(n)$	$O(n)$
Left-corner	eager	$O(1)$	$O(n)$	$O(1)$
What people do		$O(1)$	$O(n)$	$O(1)$

The table can be explained with reference to Figure 2. A top-down enumeration of the left-branching tree clearly requires storage proportional to n, the height of the tree: at the point when Z is enumerated, each of $A, B, \ldots, X$ remains incomplete because its rightmost child has not yet been encountered.[2] The same holds true for the center-embedded structure: using a top-down enumeration, each of $A, C, D, \ldots, X$ remains incomplete until the subtree it dominates has been entirely enumerated. In contrast, the top-down strategy requires only constant space for the right-branching structure: each of $A, C, \ldots, X$ becomes complete as soon as its rightmost child is enumerated, so the number of incomplete nodes at any time is at most two. We conclude that if the human sentence processing strategy were top-down, people would find increasing difficulty with both multiply left-branching and multiply center-embedded constructions, but not with right-branching constructions. The evidence exemplified by (1) suggests that this is not the case.

A similar analysis holds for the bottom-up strategy. The left-branching structure requires only constant space, since each of $X, \ldots, B, A$ becomes complete as soon as both children have been enumerated. In contrast, enumerations of the right-branching and center-embedded constructions require linear space, since every leftmost child remains incomplete until the subtree dominated by its right sibling has been entirely enumerated. The left-corner strategy with arc-standard enumeration behaves similarly to the bottom-up strategy, since every *parent* node remains incomplete until the subtree dominated by its right sibling has been

[2] Abney and Johnson discuss space complexity with respect to the length of the input string, not the height of the parse tree, but if we assume the grammar is finitely ambiguous this distinction is of no importance.

entirely enumerated. If increased memory load is responsible for increased processing difficulty, as we have been assuming, then both the bottom-up strategy and the arc-standard left-corner strategy predict that people have more difficulty with right-branching than with left-branching structures. Our conclusion is the same as for the top-down strategy: the asymmetry of the prediction is not supported by the evidence.

On the other hand, arc-eager enumeration makes a critical difference to the left-corner strategy when applied to the right-branching structure. Recall that the left-corner enumeration of nodes for this structure is BADC.... Notice that after node C has been enumerated, arc $\langle A, C\rangle$ is introduced *immediately*, and as a result, node A is no longer incomplete. In general, the arc-eager left-corner strategy will enumerate the right-branching structure with at most three nodes incomplete at any point. Furthermore, as was the case for the bottom-up strategy, the left-branching structure requires constant space. We see that only the center-embedded structure requires increased storage as the depth of embedding increases. Thus of the four strategies, the arc-eager version of the left-corner strategy is the only one that makes predictions consistent with observed behavior.

2.2 Two Problems

Under the assumptions made by Abney and Johnson, the discussion sketched out above does make a case for a left-corner strategy being more psychologically plausible than top-down or bottom-up strategies. However, there are two difficulties with the argument as it is presented.

First, by abstracting away from parsing *algorithms* and placing the focus on parsing *strategies*, Abney and Johnson make it difficult to fairly compare space requirements across different methods of parsing. Without a formal characterization of the algorithms themselves, it is not clear that their abstract notion of space utilization means the same thing in each case.[3]

For example, consider the augmented transition network (ATN) in Figure 3, where the actions on the arcs are as follows:

I1:	*np1*	←	*
I2:	*result*	←	(S (np1 *))
I3:	*det1*	←	*
I4:	*result*	←	(NP (det1 *))
I5:	*result*	←	*a*
I6:	*result*	←	*the*

Uppercase arc labels represent PUSH operations, and lowercase labels represent terminal symbols. In the pseudolanguage used here for arc actions, *np1*, *det1*, and *result* are registers, the leftward arrow (←) indicates an assignment statement, the *pop* arc transmits control (and the contents of the *result* register) to the calling subnetwork, and the asterisk (*) represents the value so transmitted (cf. [Woo70]). So, for instance, action I4 constructs an NP dominating the structure in the *det1* register on the left, and, on the right, the noun structure received on return from a push to the N subnetwork.

[3] I am grateful to Stuart Shieber for this observation.

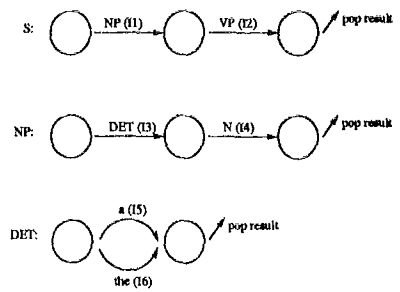


Figure 3: *Fragment of an ATN*

Now, the ATN is perhaps one of the most common examples of a parser operating in a top-down fashion. Yet according to the definitions proposed by Abney and Johnson, the enumeration performed by the ATN parser given above would seem to make it an instance of a bottom-up strategy. For example, in parsing the noun phrase *the man*, the ATN above would recognize the determiner *the*, then the noun *man*, and finally it would build and return the structure $[_{\mathrm{NP}}$*the man*$]$ from the NP subnetwork. The source of difficulty lies in the decoupling of the parser's *hypotheses* from the *structures* that it builds. When the determiner *the* is encountered, no parse-tree structures have been built, but the mechanism controlling the ATN's computation has stored the hypotheses that we were parsing an S, that we had entered the NP subnetwork, and that we had subsequently entered the DET subnetwork. These correspond precisely to the nodes we expect to see enumerated during the course of a top-down strategy.

One could, of course, choose in this case to identify the space utilization of this parser with the hypotheses rather than the structures built. However, that leaves the status of the structures themselves in question. More to the point, re-characterizing the storage requirements of a particular algorithm is exactly the sort of manipulation that the abstract notion of parsing strategies should help us avoid.

The second difficulty with Abney and Johnson's discussion concerns the distinction between arc-eager and arc-standard strategies. As they point out, for both top-down and bottom-up strategies, the two forms of arc enumeration are indistinguishable. In addition, left-corner parsing with arc-standard enumeration is, at least for the purposes of this discussion, virtually identical to bottom-up parsing, having no distinguishable effects either with respect to space utilization or even with respect to the hypotheses that are proposed.[4] So it seems somewhat odd to introduce a distinction between "eager" versus "standard" when it turns out to distinguish only one of six possible combinations (top-down/eager, top-down/standard, etc.). The question of exactly what "eager enumeration" does would seem to merit further attention. We shall give it that attention shortly, in Section 4.

3 Comparing Automata

Abney and Johnson's argument is largely an independent account quite similar to one made earlier in [JL83]. Here we present a brief summary of the argument as presented there. Johnson-Laird's presentation, though it encounters a difficulty of its own, turns out to complement Abney and Johnson's and to make clear how to solve the difficulties in both.

Following the standard description in the compilers literature (see, e.g., [ASU86]), Johnson-Laird adopts the definition of a *top-down* parser as one that operates by recursive descent: it begins with the start symbol of the grammar and successively rewrites the leftmost nonterminal until it reaches a terminal symbol or symbols that can be matched against the input. Parsing in this fashion, the parse tree is constructed top down and from left to right. A *bottom-up* parser builds the tree by working upward from the terminal symbols in the input string, constructing each parent node after all its children have been recognized. A *left-corner* parser recognizes the left-corner of a context-free rule bottom-up, and predicts the remaining symbols on the right-hand-side of the rule top-down.

Johnson-Laird examines the psychological plausibility of parsers, not parsing strategies, but otherwise his argument is very much the same as the discussion in the previous section. He concludes that the symmetry of human performance on left- and right-branching structures counts against the top-down and bottom-up parsers, and that the left-corner parser is a viable alternative that appears to be consistent with the evidence. He then provides a more formal characterization of the various parsers by expressing each as a push-down automaton (PDA). Such a characterization immediately remedies the first difficulty we found in [AJ91]: the formal specification of each parsing algorithm permits us to express space utilization uniformly in terms of the automaton's stack.

[4] Although top-down filtering can be added (see, e.g., [PS87, p. 182]), Schabes (personal communication) points out that left-corner parsing with top-down filtering is essentially the same as LR parsing. Top-down filtering restricts the non-deterministic choices made by the parser, but does not affect the bottom-up construction of the parse tree along a single computation path.

The top-down and bottom-up automata behave exactly as we would expect. The stack of the bottom-up automaton never grows beyond a constant size for left-branching constructions, but is potentially unbounded for center-embedded and right-branching constructions. The top-down automaton displays the opposite behavior, the size of its stack size being bounded only for right-branching constructions.[5]

Of particular interest is Johnson-Laird's construction of a PDA for left-corner parsing, which we consider in more detail. The stack alphabet for the left-corner PDA includes not only terminal and non-terminal symbols from the grammar, but also special symbols of the form [X Y], where X and Y are nonterminals. The first symbol in such a pair represents the top-down prediction of a node, and the second a node that has been encountered bottom-up. The use of these pairs permits a straightforward combination of left-corner prediction, which is bottom-up, and top-down prediction and matching against the input in the style of a top-down automaton.

Here we consider an extremely simple left-corner automaton, constructed from a grammar having the following productions:

(1)	S	$\rightarrow$	NP VP
(2)	NP	$\rightarrow$	John | Mary
(3)	VP	$\rightarrow$	V NP
(4)	V	$\rightarrow$	likes

The rules of the automaton are as follows:

	Input	Stack	New top of stack
1	John	...	... John
2	Mary	...	... Mary
3	likes	...	... likes
4	*ignored*	... X John	... [X NP]
5	*ignored*	... X Mary	... [X NP]
6	*ignored*	... X likes	... [X V]
7	*ignored*	... [X NP]	... [X S] VP
8	*ignored*	... [X V]	... [X VP] NP
9	*ignored*	... [X X]	...

The top of the stack is at right, and rules 4–9 are actually schemata for a set of rules in which X can be replaced by each of the nonterminals (S, NP, VP, and V). The parser begins with S on top of the stack, and a string has been successfully recognized if the stack is empty and the input exhausted.

[5] The analysis being straightforward, we omit the details here; for a complete discussion of the construction of PDAs for top-down and bottom-up parsing, see [LP81, §3.6].

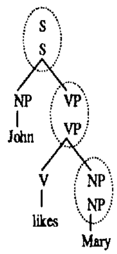


Figure 4: *Distinguishing the top-down view of a node from the bottom-up view*

Rules 1–3 simply introduce lexical items onto the stack as they are scanned. Rules 4–6 represent bottom-up reductions according to the lexical productions of the grammar (productions (2) and (4)); for example, rule 4 states that if a constituent X has been predicted top-down, and the word *John* is scanned, we continue seeking X top-down with the knowledge that we have identified an NP bottom-up. Rules 7 and 8 implement left-corner prediction: if the left-corner node of a rule has been recognized bottom-up, then we hypothesize the parent node in bottom-up fashion and also predict the right siblings top-down. For example, rule 8 states that if a V has been recognized bottom-up, we should hypothesize that a VP is being recognized and also predict the remainder of the VP, namely an NP, top-down. Finally, rule 9 pops a symbol off the top of the stack if we have predicted a constituent X top-down and then succeeded in finding it bottom-up.

In examining the behavior of this automaton for the sentence *John likes Mary*, a problem immediately becomes apparent. The contents of the stack at each step during the parse are as follows:

							Mary				
				likes		NP	NP	[NP NP]			
	John		VP	VP	[VP V]	[VP VP]	[VP VP]	[VP VP]	[VP VP]		
S	S	[S NP]	[S S]	[S S]	[S S]	[S S]	[S S]	[S S]	[S S]	[S S]	
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)

As the sentence — a right-branching structure — is recognized, we find that the stack is accumulating symbols of the form [X X]. It is clear that as the depth of right-branching increases, the number of stacked-up symbols of this form will also increase, without upper bound. Why is this happening?

Let us distinguish between the top-down "view" of a node and the bottom-up (left-corner) "view" of that node. Figure 4 makes this distinction explicit: the VP predicted top-down by the rule $S \rightarrow NP\ VP$ is distinct from the VP predicted in left-corner fashion using $VP \rightarrow V\ NP$. These are, in fact, precisely the two VPs in the symbol [VP VP]. Now, ***enumerating*** the arc between VP and S in the final parse tree is equivalent to *identifying* these two views (dotted ellipses in the figure). As long as we have not identified the two views of VP as the same node, the arc is not enumerated — and the parent S remains *incomplete* in the sense defined by Abney and Johnson. It is rule 9 in the automaton that effects this identification: popping [VP VP] amounts to recognizing that the top-down view and the bottom-up view match. Since the operation of the automaton prevents the symbol from being popped until the bottom-up view has been completed, it is clear that this automaton implements an arc-***standard*** strategy rather than an arc-eager one. Hence it is not surprising that the automaton fails to support Johnson-Laird's argument: far from being bounded, the stack of such automaton can grow without bound as the depth of right-branching increases.

4 Arc-eager Enumeration as Composition

4.1 An Easy Fix...

To summarize thus far, [AJ91] and [JL83] present two forms of the same argument, but each presentation suffers from a central shortcoming. Abney and Johnson, discussing parsing strategies rather than parsers, fail to characterize top-down, bottom-up, and left-corner parsing in a way that permits a fair comparison of space utilization. Johnson-Laird, formalizing parsers as push-down automata, provides a characterization that clearly defines the terms of the comparison, but his left-corner automaton lacks the properties needed to make the argument succeed.

Modifying the left-corner automaton so that it performs arc-eager enumeration is straightforward. As discussed toward the end of the previous section, "attachment" of a node X to its parent occurs when the symbol [X X], representing the top-down and bottom-up views of that node, is removed from the stack. In order to attach the node (i.e., enumerate the arc) eagerly, we should pop the symbol as soon as it is introduced. For the automaton in the previous section, this amounts to augmenting rule schema 8 with the rule

	Input	Stack	New top of stack
8'	*ignored*	... [VP V]	... NP

and, in general, augmenting the rules of left-corner prediction so that symbols of the form [X X] are not introduced obligatorily.

It is easy to show that the automaton, modified in this fashion, requires only a finite stack for arbitrarily

$$\frac{\langle B_1 \rightarrow \gamma_1 \cdot \rangle \ldots \langle B_k \rightarrow \gamma_k \cdot \rangle}{\langle A \rightarrow B_1 \ldots B_k \cdot \rangle} \qquad \frac{\langle A \rightarrow \cdot \ B_1 \ldots B_k \rangle}{\langle B_1 \rightarrow \cdot \gamma_1 \rangle \ldots \langle B_k \rightarrow \cdot \gamma_k \rangle}$$

Figure 5: *Inference-rule characterization of bottom-up reduction step (left) and top-down prediction step (right).*

deep left- and right-branching constructions, but requires increasing stack space for center-embedded constructions as the depth of embedding increases. Thus we have succeeded in presenting a *complete* version of the argument in [AJ91] and [JL83] in the sense that

1. top-down, bottom-up, and left-corner parsing are characterized in a formally precise way,
2. the characterizations are abstract, in the sense that the logic of the algorithms (in the form of nondeterministic push-down automata) is separated from their control (namely the control of how the automata's nondeterministic choices are made),
3. the notion of space utilization (namely stack size) is the same for each case, permitting us to make a fair comparison, and
4. the conclusion, as expected, is that top-down and bottom-up parsing both make incorrect predictions, but a form of left-corner parsing is consistent with the apparent behavior of the human sentence processing mechanism.

4.2 ...and its Implications

The import of the "fix" in the previous section is not simply that the automaton can be made to display the appropriate behavior. It is that the "arc-eager" enumeration strategy is a different (and perhaps misleading) description of a parser's ability to perform *composition* on the structures that it is building.

If we describe the parsers as sets of inference rules rather than automata,[6] the inference permitting arc-eager enumeration in the left-corner parser turns out to be a rule of composition: $A \rightarrow \alpha \cdot B$ and $B \rightarrow \beta \cdot \gamma$ can be composed to form the dotted item $A \rightarrow \beta \cdot \gamma$. For instance, the effect of rule 8' is to predict $VP \rightarrow V \cdot NP$ from V, and then immediately compose this new item with $S \rightarrow NP \cdot VP$. Equivalently, the rule first predicts the VP structure in Figure 4 from the V (giving us [VP VP], corresponding to the two VP nodes the figure), and then immediately identifies the lower VP node with the upper one (which removes [VP VP]), leaving just an S structure that lacks an NP.

[6]Two descriptions that are formally equivalent.

In contrast, even if one were to add a rule of composition to the inferential description of top-down and bottom-up parsers, it would have no effect. Neither the top-down nor the bottom-up parser ever ***introduces*** a configuration in which the A constituent and B constituent are both only partially completed (and thus can be composed). Instead, these parsers rewrite the entire right-hand side of a rule at once (see Figure 5). In order for a rule of composition to be relevant, it is necessary that the parser introduce both the top-down view of a constituent (e.g. B in $A \rightarrow \alpha \cdot B$) and the bottom-up view of that constituent (e.g. B in $B \rightarrow \beta \cdot \gamma$) so that they may later be identified. Unlike top-down and bottom-up parsers, a left-corner parser meets this criterion.

By presenting a complete version of the argument in [AJ91] and [JL83], we have essentially re-discovered proposals made by Pulman [Pul85, Pul86] and Thompson *et al.* [TDL91]. Both propose parsers with left-corner prediction and a composition operation added. Pulman motivates his parser's design on grounds of psychological plausibility, though he does not present a complete version of the argument discussed here. Thompson *et al.* are motivated by issues in parallel parsing. In addition, we should note that Johnson-Laird introduces a parser with a composition-like operation later in his discussion, though outside the context of a formal comparison among parsing methods.

Abney (personal communication) points out that, though psychologically plausible in terms of the space utilization argument we have discussed, the automaton presented here may nonetheless fail to be plausible because of its behavior with regard to local ambiguity. If we opt to compose whenever possible (e.g., always preferring rule 8' to rule 8 when X = VP), which seems natural, then left-recursive structures will lead to counterintuitive results — for example, in processing (2), the automaton will prefer to attach the NP *the cat* as the object of the verb, rather than waiting for the full NP *the cat's dinner*.

2 John prepared [[the cat]'s dinner].

More generally, as Abney and Johnson discuss, there is a tradeoff between storage, which is conserved by strategies that perform attachment "eagerly," and ambiguity, which is avoided by deferring attachment until more information is present to resolve it. On the basis of the observations we have made here, it appears that this tradeoff is expressed most naturally not in terms of a comparison between different parsing strategies, but rather in terms of the criteria for when to ***invoke*** a composition operation that is available to the parser.

5 Conclusions

In this paper, we have considered a space-utilization argument concerning the psychological plausibility of different parsing methods. Both [AJ91] and [JL83] make the same basic claim, namely that top-down and bottom-up parsing lead to incorrect predictions of asymmetry in human processing — predictions that can be avoided by utilizing a left-corner strategy. We have demonstrated difficulties with both of their formulations and presented a more precise account. In so doing, we have found that composition, rather than left-corner prediction *per se*, plays the central role in distinguishing parsing methods.

In making the argument, we were forced to abandon the abstract characterization of parsing methods in terms of strategies, and return to defining parsers in terms of their realizations as automata. This has the unfortunate consequence of tying the argument to context-free grammars, losing the attractive formalism-independent quality evoked in [AJ91].

Since context-free grammars are no longer generally considered likely models for natural language in the general case [Shi85], one wonders how the discussion here might be extended to parsing within more powerful grammatical frameworks. It is interesting to note the relationship between the style of left-corner parsing described here and one such framework, combinatory categorial grammar (CCG) [Ste90]. Composition is an integral part of CCG, as is the notion of type-raising, which resembles left-corner prediction.[7] The operation of a left-corner parser with composition can fairly be described as being in the style of CCG, but retaining the context-free base. Since one attractive feature of CCG is its inherent left-to-right, word-by-word incrementality, it is perhaps not surprising to find that parsers of CCG tend naturally to meet the criteria for psychological plausibility discussed here.

CCG is one instance of a general class known as the mildly context-sensitive grammar formalisms [JVSW88]. We are currently investigating a generalization of the argument presented here to other formalisms within that class.

Acknowledgements

This research was supported by the following grants: ARO DAAL 03-89-C-0031, DARPA N00014-90-J-1863, NSF IRI 90-16592, and Ben Franklin 91S.3078C-1. I would like to thank Steve Abney, Mark Johnson, Aravind Joshi, Yves Schabes, Stuart Shieber, and members of the CLiFF group at Penn for their helpful discussion and criticism.

[7]For example, NP can be type-raised to S/(S\NP), which roughly corresponds to $S \rightarrow NP \cdot VP$.

References

[AJ91] Steven Abney and Mark Johnson. Memory requirements and local ambiguities for parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250, 1991.

[ASU86] Alfred Aho, Ravi Sethi, and Jeffrey Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 1986.

[DJK+82] A. DeRoeck, R. Johnson, M. King, M. Rosner, G. Sampson, and N. Varile. A myth about centre-embedding. *Lingua*, 58:327–340, 1982.

[JL83] Philip N. Johnson-Laird. *Mental Models*. Harvard University Press, 1983.

[JVSW88] A. K. Joshi, K. Vijay-Shanker, and D. J. Weir. The convergence of mildly context-sensitive grammatical formalisms. In P. Sells and T. Wasow, editors, *Processing of Linguistic Structure*. MIT Press, Cambridge, MA, 1988.

[Kim73] J. Kimball. Seven principles of surface-structure parsing in natural language. *Cognition*, 2:15–47, 1973.

[LP81] Harry Lewis and Christos Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

[MC63] George Miller and Noam Chomsky. Finitary models of language users. In R. Luce, R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology, Volume 2*. John Wiley, 1963.

[MI64] G. A. Miller and S. Isard. Free recall of self-embedded English sentences. *Information and Control*, 7:292—303, 1964.

[PS87] Fernando C. N. Pereira and Stuart M. Shieber. *Prolog and Natural Language Analysis*. Center for the Study of Language and Information, 1987.

[Pul85] Stephen Pulman. A parser that doesn't. In *Proceedings of the 2nd European ACL*, pages 128–135, 1985.

[Pul86] Stephen Pulman. Grammars, parsers, and memory limitations. *Language and Cognitive Processes*, 1(3):197–225, 1986.

[Shi85] S. M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343, 1985.

[Ste90] Mark Steedman. Gapping as constituent coordination. *Linguistics and Philosophy*, 13:207–263, April 1990.

[TDL91] H. Thompson, M. Dixon, and J. Lamping. Compose-reduce parsing. In *Proceedings of the 29th Annual Meeting of the ACL*, pages 87–97, June 1991.

[Woo70] William A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, October 1970.