A Computational Model of Human Parsing

Steven P. Abney
Bell Communications Research
Morristown, NJ 07960-1910

Published in:
The Journal of Psycholinguistic Research. Vol. 18. No. 1

A Computational Model of Human Parsing

1. Introduction

This report describes a computer program which is intended as a computational model of human parsing. Two aspects of human parsing receive particular attention: (1) the role of attachment preferences, and (2) the distinction between *weak* and *strong* garden paths -- strong garden paths being irrecoverable, weak garden paths causing psychological difficulty, but not preventing recovery of the correct structure. The program I describe is intended as an explicit hypothesis about the mechanisms underlying human syntactic analysis; though of course with the caveat that it characterizes human cognitive processes only at a certain level of abstraction.

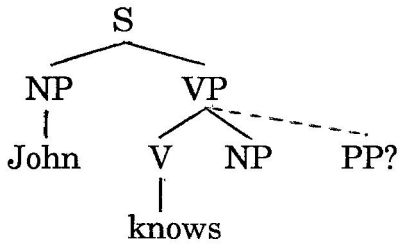
I will assume without argument that the human parser has the following properties: it is deterministic on most input it encounters; when it does parse non-deterministically, it backtracks, as opposed to pursuing multiple parses in parallel; and the parser's backtracking is experimentally detectable. For argumentation on these points, I refer the reader to the work cited in the bibliography by Lyn Frazier and her collaborators.

1.1 Standard Deterministic Parsers

Most natural language parsers are thoroughly non-deterministic, because of the high degree of local ambiguity in natural language. There have been preliminary attempts at adapting deterministic parsers to natural

language by providing them with human-like preferences for resolving local ambiguities (Shieber, 1983; Pereira, 1985). But standard deterministic parsers are not well-suited to psycholinguistic modelling. Shieber and Pereira proposed supplementing a standard bottom-up deterministic parser (i.e., an LR parser) with conflict-resolution strategies to model human parsing preferences. However, LR parsers cannot model the incrementality of human parsing. Namely, in right-branching structures, LR parsers build no structure until all the input has been read. LR parsers build such structures literally bottom-up. The lowest node is the first node built, and it is not built until all the input has been read. By contrast, people clearly build structure before the end of the sentence, and pass it incrementally to the semantic processor.

Standard top-down deterministic parsers (e.g., LL parsers) are capable of producing structures incrementally. Unfortunately, though, they introduce spurious local ambiguities which force them to backtrack where people do not. Consider, for example, a grammar which includes the rules VP --> V NP PP and VP --> V NP. On a sentence beginning "John found", the parser builds the following structure, and attempts to expand VP:



However, the parser has insufficient information to determine which expansion to use. Therefore, it must guess, and backtrack if it guesses

wrong. This means it will be forced to backtrack on one of "John found the book" or "John found the book on the desk"; this is unsatisfactory, since there is no indication that people backtrack on either of these sentences.

1.2 Licensing-Structure Parsing

The parser I will consider here, a *licensing-structure* (LS) parser, does not suffer from either of these drawbacks. Intuitively, the common problem with LR and LL parsing is that they use global characterizations of phrasal expansions. In LR parsing, for example, a node can be built only when *all* daughters of the node have been found; and in LL parsing, when a node is expanded, the parser must decide on the identity of *all* the daughters of the node at once. In licensing-structure parsing, by contrast, node expansions are collections of smaller pieces of structure, viz., binary relations between head and sister -- a sister appearing in the expansion of XP iff it is assigned a *licensing relation* by the head of XP. Unlike in LR parsing, we can build a VP as soon as we have seen the V, without waiting to see the remaining daughters of VP; but unlike in LL parsing, we do not have to specify what those daughters are going to be at the point of VP-expansion. We can incorporate them as they appear, whether only an NP follows, or both an NP and a PP.

A licensing "grammar" consists in a set of specifications of the licensing relations various words (or classes of words) can assign. The following are examples of phrase structure rules and the corresponding licensing-relation sets:

(2) VP --> V_{kiss} NP (PP)
kiss: [--> NP θ],
[--> PP M]

(3) S --> NP VP
$[_I \varepsilon]$: [<-- NP S],
[--> VP F]

The licensing relation [--> NP θ], for example, specifies that *kiss* can license a noun phrase to its right by θ-assignment (i.e., as an argument). In (3), we are forced to assume an empty head for the exocentric category S. ("I" or "Infl" stands for "Inflection": see e.g. (Chomsky, 1986).) All four relation types are illustrated in this small example: θ-assignment (θ), modification (M), subjecthood (S), and functional selection (F). The first three are self-explanatory. *Functional selection*, for want of a standard term, is what I call the relation between a functional ("non-lexical", "extragrammatical") category like Infl and its complement (VP, in the case of Infl).

There is a natural method for parsing licensing grammars, the simplest version of which is as follows. A "Simple Licensing-Structure parser" (SLS-parser) is a non-deterministic parser which recovers licensing structure by considering pairs of nodes, attaching one to the other where possible. More precisely, a configuration of an SLS parser consists of a stack of unattached nodes, and the unparsed input (see figure 1a). The top of the stack is to the right. Two of the partially-assembled trees are distinguished. The tree on top of the stack (furthest to the right) is the current tree (C), and the tree immediately below it (to its left) is the left tree (L).

(Insert figure 1 about here.)

At any point, there are three actions the parser can take: (1) it can Shift a word from the input onto the stack, projecting it to XP (figure 1b); (2) it can Attach the current tree as a right sister to some head in the near edge of the left tree (figure 1c) -- provided that the head in question has a right-directional relation by which it can license the current node; or (3) it can perform the mirror-image operation, which I call Attach-L (figure 1d), provided the head in the near edge of the current tree has a left-directional relation by which it can license the left node. ("Attach-L" stands for "Attach the Left node". For consistency's sake, we can consider "Attach" to be an abbreviation for "Attach-C".)

A sample parse is presented in figure 2.

(Insert figure 2 about here.)

2. Conflict Resolution

The next action of an SLS parser will be highly under-determined. In accordance with our assumptions about the nature of the human parser, then, the first step in making an SLS parser into a psycholinguistic model is to "determinize" it. We require a set of conflict-resolution strategies, which determine which action to choose when more than one next action is legal.

There are six logically possible next-action conflicts:

1. Attach/Attach
2. Shift/Attach
3. Shift/Attach-L
4. Attach-L/Attach-L
5. Shift/Shift

6. Attach/Attach-L

I will discuss each in turn.

2.1. Attach/Attach Conflicts

Attach/Attach conflicts arise in the examples like *I saw the man with the telescope*. At the point where *with the telescope* is the current tree, and *I saw the man* is the left tree, there are two ways of performing an Attach action: the PP may be attached as a right sister to the N or to the V.

I propose that Attach/Attach conflicts be resolved in favor of θ-attachment -- i.e., attachment mediated by a θ-relation -- and failing that, in favor of attachment to a canonical θ-assigner, i.e., a verb. In detail, I propose the following preferences:

P1: Prefer θ-attachments over non-θ-attachments.
P2: Prefer attachment to verbs over attachment to non-verbs.
P3: Prefer low attachment.

These strategies are applied in the order given. The following examples illustrate their application:

(4) a. Prefer θ-attachment

i. I saw a man ⟍ with a telescope

ii. I thought about his interest —→ in the Volvo

iii. the destruction of the city ⟍ with a 20M-ton warhead

b. Else, prefer V

i. I sang to the cat — in the kitchen

ii. I wrote a letter — to a friend

c. Else, attach low

a gift to a boy — in a box

(The last example, by the way, underlines the fact that attachment preferences are genuinely *syntactic* preferences, not semantic preferences: the preferred analysis in (c) is semantically anomalous.)

This set of preferences is intended as an alternative to Frazier's Minimal Attachment and Late Closure. I have serious misgivings about Minimal Attachment on both empirical and theoretical grounds. Section 3 provides a discussion of the issues.

As for Late Closure, because I am approaching parsing preferences in terms of next-action conflict resolution, rather than in terms of structural properties of competing analyses, it is necessary to revert to something similar to Kimball's original "Right Association" and "Closure" principles (Kimball 1973). P3 (Low Attachment) corresponds roughly to Kimball's Right Association principle. Kimball's Closure principle corresponds in the present model to P4 below.

2.2. Shift/Attach, Shift/Attach-L Conflicts

The general principle determing the resolution of Shift/Attach and Shift/Attach-L conflicts is "if you can build structure now, do so". In other words, Shift is an action of last resort:

P4. Prefer Attach over Shift

P5. Prefer Attach-L over Shift

2.3. Attach-L/Attach-L Conflicts

A consequence of resolving Shift/Attach-L conflicts in favor of Attach-L is that Attach-L/Attach-L conflicts never arise. In an Attach-L/Attach-L conflict, the Left node (L) can be licensed by two heads in the near edge of the current tree, call them A and B. At an earlier point in the parse, only L and *one* of A and B were on the stack. At that point, we had a Shift/Attach-L conflict. Resolving Shift/Attach-L conflicts in favor of Attach-L effectively resolves potential Attach-L/Attach-L conflicts in favor of low attachment.

This appears to make the right predictions, empirically. Consider examples like:

(5) a. [quite surprisingly] big (low)
b. [quite surprisingly big] (high)

(6) a. those [hundred pound] bags (low)
b. those [hundred pound bags] (high)

(5a) means "big to a quite surprising degree", (5b) means "quite big, and surprisingly big". (5a) seems to be the prefered interpretation. (6a) and (6b) are adapted from (Marcus, 1980). In (6a), the bags each weigh 100 pounds; in (6b) the bags weigh a pound, and there are 100 of them. Again, the low-attachment structure (6a) is prefered.

2.4. Shift/Shift Conflicts

A Shift-Shift conflict arises exactly when the next word is categorially ambiguous: the conflict being which category is to be shifted onto the stack. I will not hazard a proposal about the resolution of lexical ambiguity, beyond suggesting that the simple expedient of ranking alternatives in the lexicon will take us a good way. There are quite complex issues involved, though, that I cannot hope to do justice to here.

2.5. Attach/Attach-L Conflicts

A final type of conflict which could arise in theory is Attach/Attach-L. For an Attach/Attach-L conflict to arise, both L and C must be able to license the other. I have been unable to construct any examples where that occurs.

2.6. Conclusion

To sum up, we require basically three preferences:

1. Prefer θ-attachment, or failing that, attachment to a canonical θ-assigner
2. As a default, prefer low attachment
3. Shift only as a last resort

These resolve all conceivable conflicts, with the exception of Shift/Shift conflicts, about which I have little insightful to say, and Attach/Attach-L conflicts, which, though conceivable, appear not to arise in fact.

3. On Minimal Attachment

In this section, I would like to consider Frazier's Minimal Attachment principle (Frazier, 1978):

> Minimal Attachment (MA): Choose the analysis which involves building the least number of nodes

Minimal Attachment (and its companion, Late Closure) have been widely adopted in the literature. However, I have felt it necessary to make the alternative proposal presented in the previous section, for a number of reasons.

3.1. Empirical Predictions

First, MA makes empirical predictions that disagree with the attachments assumed in the sentences of (4). For example, MA predicts that PP-attachment to NP will never be prefered over attachment to VP -- Frazier assumes that PP's are uniformly Chomsky-adjoined to NP, but simply attached to VP, hence that it takes one more node to associate a PP with NP than to associate it with VP. I claim that attachment to NP will be prefered when the noun θ-marks the PP, but the verb does not. The crucial example is sentence (4.a.ii.), "I thought about his interest in the Volvo". My intuitions are strong that attachment to NP is prefered -- though of course final resolution of the issue waits on experimentation.

Consider also cases where two NP's are involved. MA makes no predictions; hence Late Closure takes over, predicting uniformly low attachment. On the other hand, my strategies predict high attachment when the lower attachment is a non-θ-attachment, and the higher

attachment is a θ-attachment. The crucial examples, then, are ones like (4.a.iii.), *the destruction of the city with a 20M-ton warhead.* My strategies predict high attachment ("destruction by means of a warhead"); Frazier's strategies predict low attachment ("the city that possesses a warhead"). Again, my intuitions are strong that high attachment is empirically the correct preference.

3.2. Structural Assumptions

My second reservation concerning MA is that it depends essentially on certain structural assumptions that I find linguistically unjustifiable. Frazier assumes that PP's uniformly Chomsky-adjoin to NP's, requiring the creation of a new NP node, whereas they simply attach to VP's, requiring no extra node. This assumption has not been defended on linguistic grounds, to my knowledge, and I consider it highly implausible.

A position that *would* be defensible is that adjunct PP's are Chomsky-adjoined, while argument PP's are simply attached. In conjunction with this position, Minimal Attachment would make roughly the predictions of P1 above, without P2. Hence it would make no prediction in cases where both sites or neither site involves argument-attachment (examples (4.b.)), with the result that Late Closure takes over, predicting low attachment -- exactly the wrong prediction.

If we make what is perhaps the standard structural assumption -- that base-generated PP's are never Chomsky-adjoined -- then there is no distinction between attachment to VP and attachment to NP in terms of number of nodes built, hence MA makes no predictions at all. Late Closure

takes over, predicting low attachment, and the result is that PP's are uniformly attached to NP's, a hopelessly wrong prediction.

3.3. Parsing Assumptions

My third qualm with MA is that it depends crucially on contestable assumptions about the order in which nodes are built. Consider the fact that utterances are frequently not complete sentences, but PP's or NP's standing alone. It would be reasonable, then, if the parser did not build an S node until it had evidence that it was in fact dealing with a complete sentence, i.e., until it encountered a word that could not be incorporated into the NP it was building. Under this view, in the main clause analysis of *the horse raced past the barn fell*, the parser builds both VP and S when it encounters *raced*. If we assume that the "reduced relative" reading involves building only a PtcP over *raced* (rather than a true relative clause), and if we assume this PtcP attaches directly under NP, rather than Chomsky-adjoining, then PtcP is the only node built under the reduced relative analysis -- versus two nodes, VP and S, under the main clause reading. Given these assumptions, MA falsely predicts that the reduced relative reading is prefered.

In short, MA only works if one makes a number of auxiliary assumptions which have not been defended, to my knowledge. Certain of these assumptions I cannot accept -- in particular, Chomsky-adjunction to NP but not to VP, and the assumption that S is built as soon as the first word is encountered. Hence, I am unable to adopt MA.

4. State

Any parser which recovers a unique initial parse will necessarily misparse certain sentences, initially. We desire that those sentences be precisely those which people initially misparse; it is the role of the conflict-resolution preferences we have discussed to guarantee that this is the case. But we also desire that the parser be able to recover from misparses in precisely those cases where people are able to recover. In the remainder of the paper, I discuss when and how the parser backtracks from an initial misparse.

4.1. LR-States

First, the parser must be able to recognize that it has misparsed. A simple LS-parser is not able to recognize that it has made an error, at least not until it reaches the end of the input and discovers that it has multiple trees on the stack. This is so because Shift is always a legal action. We must permit the parser to shift when it cannot attach, because in left-branching structures, the licenser of the current tree is still waiting in the input. What we need, then, is an encoding of local state, such that the parser knows when it is conceivable that a licenser is yet to come, and when no following word could possibly rescue the current partial structure.

I will use standard techniques to accomplish this. A *state* in LR parsing is a characterization of how much of a phrasal expansion one has parsed. For example, the state

(7) [NP --> Det . AP N]

encodes the following information: (a) the parser is building an NP, (b) it has seen a Det, and (c) it is expecting an AP next, followed by an N. On any word that can start an AP, state (7) has what I will call a *continuation*. For example, if the next word is A, the continuation is the state [AP --> A .], i.e., we are now building an AP, and we have seen an A. As long as there is a continuation on the next word in the input, the parser may Shift. If there is no continuation on the next word -- if the next word is a verb, for example -- the parser recognizes that it has gone down a garden path: either it made an error at some choice point, or the sentence is ill-formed.

When we reach the end of an expansion encoded in an LR-state, we perform a "reduction", which consists of a series of Attach-L actions. For example, when we reach the state [NP --> Det AP N .], N is on the top of the stack. We project it to NP, and do two Attach-L's to incorporate the AP and Det preceding it.

In brief, then, LR-states keep track of unassembled pieces of a phrasal expansion, and encode information about which continuations will lead to a well-formed phrase. They are necessary only until the licensing head has been found -- i.e., on left-branching portions of the structure. On right-branching structures, words can be immediately assimilated as they are shifted onto the stack; hence there is no need for local state to keep track of the pieces of an unassembled phrase.

Figure 3 illustrates the use of LR-states in an LS-parse. For reasons that will become clear as we proceed, we show the stack explictly, by connecting members of the stack with dotted lines. We include LR-states on the stack for ease of recognizing which is the current state (viz., the one on top of the stack), but they are otherwise "invisible". The next action in figure 3 will be

a reduction, i.e., a sequence of two Attach-L actions, which will assemble the NP.

(Insert figure 3 about here.)

An exception to the rule that LR-states are used only on left-branching pieces of structure arises with functional categories like Infl. Infl is frequently phonologically empty, presenting a problem for recognizing when it has been encountered. Let a licensing grammar's *LR-relations* be the class of relations used to construct LR states. A solution to the empty-head problem is to include left-directional relations *plus* functional-selection relations among the LR-relations.

For example, instead of using the LR-state [IP --> . NP (I)], we will use the state [IP --> . NP (I) VP]. We will know to build an IP when we have seen NP and VP, even if the Infl is phonologically null. (If the Infl is null, we will insert it when we do the reduction.)

This contrasts with the LR-state used in parsing VP, which will not include any complements of V, inasmuch as complements of V are not functionally-selected, but are either θ-complements or modifiers. Correspondingly, V must be overt; we cannot permit empty V's.[1]

[1]Taking this approach to the problem of empty heads requires us to put strong constraints on functional selection. If we allowed V to functionally-select IP, for example, we would introduce into LS-parsing the problem which caused us to reject LR-parsing: namely, that in certain right-branching structures, no structure is built until the end of the input. Basically, we cannot permit functional-selection cycles of the form

$[_{VP}$ X --F--> $[_{VP}$ Y ... --F--> $[_{VP}$ ...

4.2. LS-States

For consistency's sake, we can construct a kind of state from right-directional relations as well. For example, if V licenses a noun phrase to its right, we can map the relation [--> NP θ] to an "LS" state

(8) [--> . NP]

(8) signifies that we have seen nothing, but we are looking for an NP. Like an LR-state, it has a continuation on any word which can start an NP. But unlike an LR-state, when we have reached the end of an LS-state (e.g., when we have seen an NP and gone on to the state [--> NP .]) we do not do a reduction, but an attachment. To keep track of which node to attach to, we start a new stack for every right-directional relation, where the licenser possessing the relation to assign is the initial node on the stack. Figure 4 illustrates. We have two stacks -- an "LR-stack", terminating in the LR-state [IP --> NP . (I) VP], and an "LS-stack", terminating in the LS-state [--> . PP]. The LS-state has a continuation on the next word, *in*. Eventually, a PP is built, and the current state becomes [--> PP .], which calls for attachment of the PP as a right sister to the N.

(Insert figure 4 about here.)

5. Backtracking

Let us consider now how this machinery will permit us to recognize misparses, and how the parser recovers from misparses. At a choice point

-- i.e., where there is more than one legal continuation -- the parser chooses one continuation on the basis of the conflict-resolution preferences discussed earlier. The other continuations are abandoned. Backtracking consists in finding an abandoned continuation and "resurrecting" it -- i.e., making it the current LR-state. We continue parsing from the new state, ultimately recovering an alternative parse. In recovering the second parse, we can reuse a good deal of the structure built during the first parse -- in fact we can do so without destroying the initial parse. For reasons of space, though, I will not discuss how this is done.

The key question is how the parser finds abandoned continuations, and decides which to resurrect. I propose that there are a handful of simple heuristics for finding abandoned continuations, and the difference between weak and strong garden paths is that the heuristics that would be necessary to find the relevant continuation in the case of strong GP's are too "complex", in an intuitive sense of *complex* which will suffice for the clear cases I wish to discuss here.

Let us contrast two well-known examples, one a weak garden path, and one a strong garden path. Consider first the weak garden path, "While she was mending the sock fell off her lap." The crucial choice point comes in the configuration of figure 5a. Both the LR state [IP --> CP . NP VP] and the LS state [--> . NP] have continuations on NP; state (1) for the former, state (2) for the latter. Continuation (1) calls for a shift, and continuation (2) calls for attachment; attachment is preferred over shifting, so continuation (2) is chosen as the new current state, and continuation (1) is abandoned.

(Insert figure 5 about here.)

The parse continues, until we reach the configuration in figure 5b. In figure 5b, the current state, [IP --> CP . NP VP], has no continuation on the verb *fell*. Hence, the parser recognizes that it has misparsed, and must backtrack. (The state given in parentheses is the abandoned continuation (1) of figure 5a.)

At this point, we require a heuristic for finding the relevant abandoned continuation. A candidate is the following:

> Search the right edge of the Left tree for an abandoned state which has a continuation on the Current node.

This heuristic will find the abandoned state [IP --> CP NP . VP] in figure 5b. Making this the current state will ultimately yield the correct analysis, in which *the sock* is the subject of the main clause.

Let us contrast this with the parser's actions on a strong garden path, *the boat floated sank*. Let us assume a very simple structural analysis, in which the conflict is between a postnominal participle phrase, and a matrix verb. At the point of conflict, the structure is that given in figure 6a. We have a Shift/Shift conflict. Assuming that the verb reading is prefered over the participle reading, continuation (1) becomes the new LR-state, and (2) is abandoned.

(Insert figure 6 about here.)

At the point of backtracking, we have the situation in figure 6b. The current state, [--> . PP], has no continuation on V. If we search the right edge of the left tree, we find no relevant abandoned states. Even if we look at

lexical alternatives, and find the abandoned state (2), state (2) has no continuation on V either. The chain of deductions that would be necessary to determine that state (2) is relevant would go something like this: if we build the PtcP and attach it to N, we can get *floated* "out of the way", and the current state will become [IP --> NP . VP], which *does* have a continuation on V. In effect, to determine if state (2) is relevant, the parser has to blindly resurrect it and see what happens. My claim is that weak garden paths arise where an abandoned state is both accessible and clearly relevant; the parser fails to find the correct alternative parse otherwise.

6. Conclusion

In sum, I have proposed a particular way of modelling human parsing. First, we require a basic parsing algorithm which provides us with the right "traces" in the default case -- i.e., which builds nodes at the right time. (Thus, for example, we rejected standard LR parsing, because nodes were built much too late in right-branching structures.) Second, there is a set of preferences for local ambiguity resolution. I believe the preferences I have proposed are empirically more adequate than Frazier's proposals, and are not dependent on questionable assumptions about syntactic structure. Finally, there are heuristics for deciding which parse to pursue next at the point of backtracking. If we suppose that the parser only has a restricted set of "simple" heuristics -- on an intuitive notion of "simplicity" that is at least adequate for certain clear cases -- then we can give an account of the difference between strong and weak garden path sentences.

Bibliography

Abney, Steven (1987). Licensing and Parsing. Proceedings of NELS 17, University of Massachusetts, Amherst, Mass.

Aho, Alfred V., and S.C. Johnson (1974). LR parsing. Computing Surveys, 6(2):99-124.

Aho, Alfred V., and S.C. Johnson (1975). Deterministic Parsing of Ambiguous Grammars. Communications of the Association for Computing Machinery, 18(8):441-452.

Chomsky, Noam (1986). Knowledge of Language. Praeger Publishers.

Frazier, Lyn (1978). On Comprehending Sentences: Syntactic Parsing Strategies. Unpublished doctoral dissertation, University of Connecticutt, Storrs, Connecticutt.

Frazier, Lyn, and Janet Fodor (1978). The Sausage Machine: a new two-stage parsing model. Cognition 6:291-325.

Frazier, Lyn, and Keith Rayner (1982). Making and Correcting Errors during Sentence Comprehension: Eye Movements in the Analysis of Structurally Ambiguous Sentences. Cognitive Psychology 14:178-210.

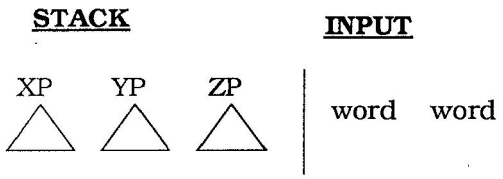
Kimball, John (1973). Seven principles of surface structure parsing in natural language. Cognition 2(1):15-47.

Marcus, Mitchell (1980). A theory of syntactic recognition for natural language. Cambridge, Massachusetts: MIT Press.

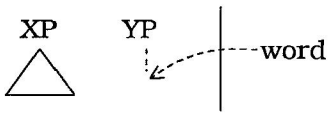
Pereira, Fernando C.N. (1985). A new characterization of attachment preferences. In: David Dowty et al., eds., Natural language parsing New York: Cambridge University Press, pp. 307-319.

Shieber, Stuart (1983). Sentence disambiguation by a shift-reduce parsing technique. In: Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, pp. 113-118.

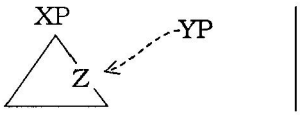
Figure 1: An LS Parser



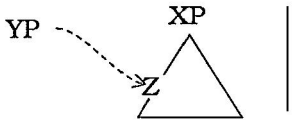
(a) *A Snapshot of the parser*



(b) *Shift*



(c) *Attach*



(d) *Attach-L*

Figure 2: A Parse

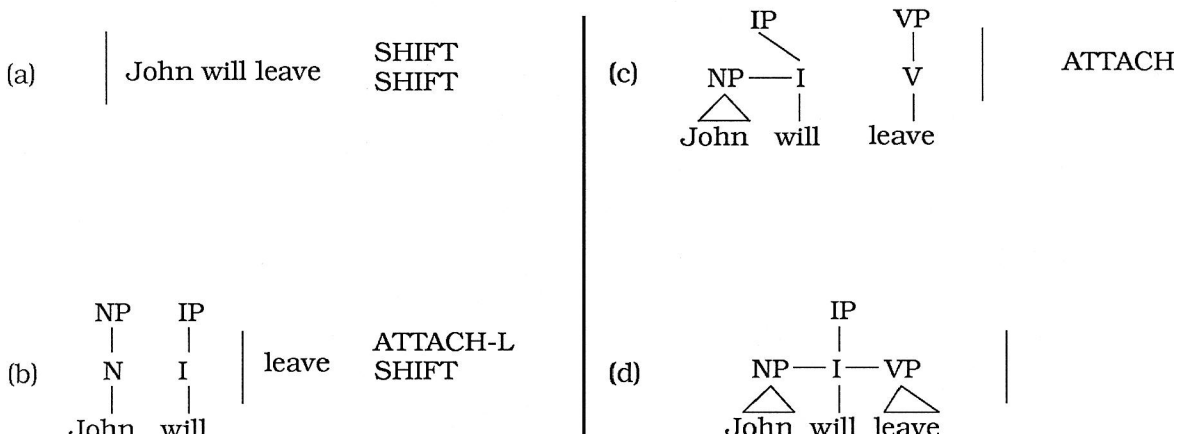


Figure 3: LR-states in an LS parse

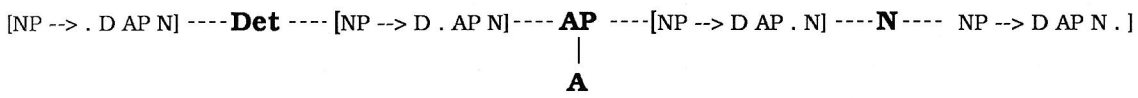


Figure 4: LR and LS states

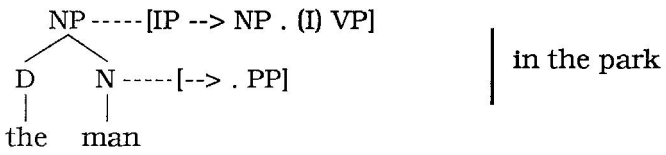
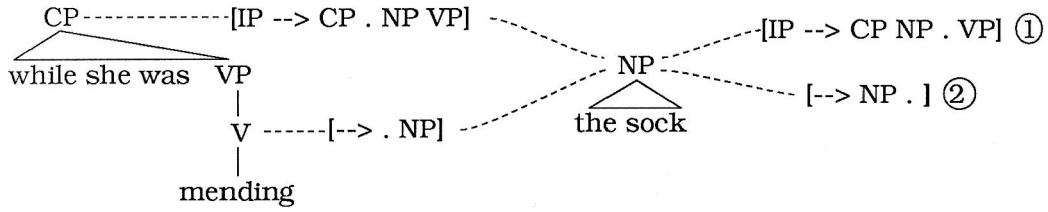
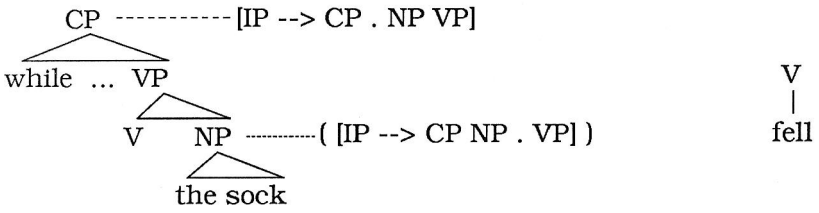


Figure 5: A Weak Garden Path

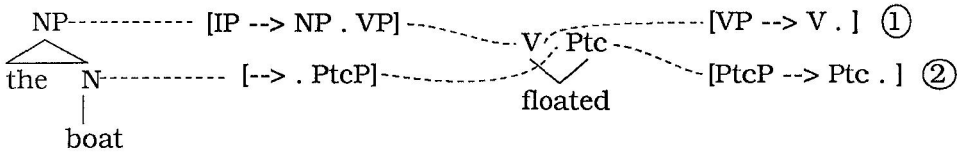


(a) *At the point of conflict*

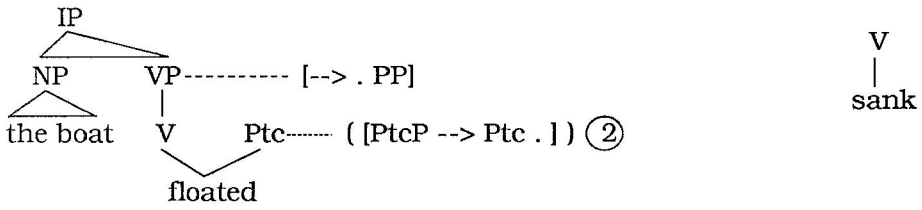


(b) *At the point of backtracking*

Figure 6: A Strong Garden Path



(a) *At the point of conflict*



(b) *At the point of backtracking*