A Parser That Doesn't

S. G. Pulman
University of Cambridge
Computer Laboratory
Corn Exchange Street
Cambridge CB2 3QG, UK.

Abstract

This paper describes an implemented parser-interpreter which is intended as an abstract formal model of part of the process of sentence comprehension. It is illustrated here for Phrase Structure Grammars with a translation into a familiar type of logical form, although the general principles are intended to apply to any grammatical theory sharing certain basic assumptions, which are discussed in the paper. The procedure allows for incremental semantic interpretation as a sentence is parsed, and provides a principled explanation for some familiar observations concerning properties of deeply recursive constructions.

Background

The starting point for the present work is a set of familiar and, for the most part, uncontroversial claims about the nature of grammatical description and of human parsing of natural language. These claims and assumptions can be briefly summarised as follows:

A Hierarchical Structure

Linguists assign constituent structures to sentences on the basis of distributional tests of various kinds. On the basis of these tests, the 'correct' structures are always hierarchical and often deeply nested. The tree representing a sentence may impose a great deal of structure on it, with string-adjacent items often appearing at very different levels in the tree. In general, shallow, 'flat' structures are not generated by grammars, nor warranted on distributional grounds. However, as we shall see, it is likely that these deeply nested structures may be somewhat remote from any that are actually computed during parsing.

B Semantics is (i) compositional and (ii) syntax-driven.

Both of these claims can be made in a variety of versions of different strengths, from the trivially true to the fairly clearly false. What is intended here is the assumption sometimes called the 'rule to rule' hypothesis, shared by almost all current grammatical frameworks, that to each syntactic rule of a grammar (or for each subtree induced by such a rule) there is an associated semantic rule, either producing an interpretation directly, or translating into some formal language. Interpretations for whole sentences are built up from the constituent parts in ways specified by these rules, in a fashion which mimics and uses the syntactic structure of the sentence.

C Incremental interpretation

As a sentence is parsed, its interpretation is built up word by word: there is little or no delay in interpreting it. In particular, we do not wait until all syntactic constituents have been completed before beginning to integrate then into some non-syntactic representation. Ample intuitive and experimental evidence supports this uncontroversial observation.

D Limited recursion.

One of the most firmly established facts about human syntactic processing is that constructions which are ineliminably deeply recursive (such as central self-embeddings) are difficult or impossible to parse. A sentence like:

1 The boy who the girl that the dog bit liked ran away

is clumsy at best, and one like:

2 The boy the girl the dog the cat scratched bit saw left

is utterly unmanageable under normal circumstances.

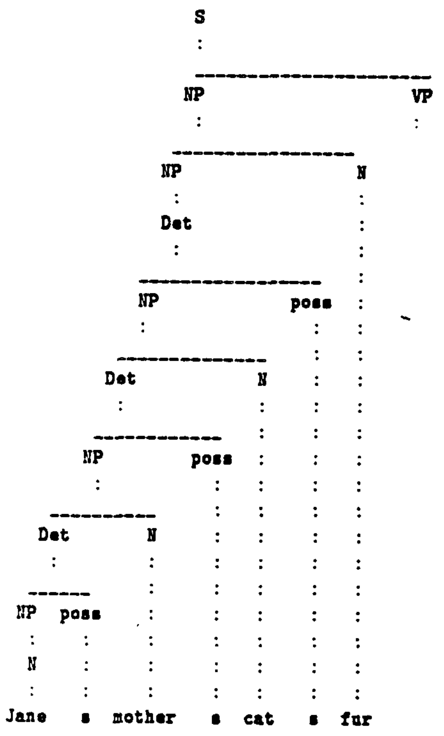
Under the further assumption, recently more controversial (Katz 1981), that grammars have some kind of mental reality as representations of linguistic knowledge, it is clear that A to D, although simple and generally agreed upon observations, by no means obviously consistent with each other. Consider, for example, the natural way in which one might set about implementing a system which observed B, a principle which, in itself, is a computationally natural principle. Such a system might first parse a sentence, annotating nodes in the resulting tree with an indication of the syntactic rules used. This annotated tree would then be passed to an interpretation routine which applied the appropriate semantic operation to the topmost node (guided by the syntactic information found there, in particular a pointer to the semantic information necessary), calling itself recursively on each subtree to build up the complete interpretation. (Systems operating in more or less this manner are described in Rosenschein and Shieber 1982, Gawron et al. 1982 and Schubert and Pelletier 1982. They are not intended as psychological models in any but the most abstract sense, of course.)

Such a system would, in observing B, also naturally be consistent with A. Obviously, though, this type of system requires a complete syntactic analysis to be available before it can even begin the process of interpretation, thus conflicting straightforwardly with C.

Consider next A and D. The structures which linguists postulate in accordance with A are often recursive, and it is in the nature of hierarchical structures that this should be a possibility. This is rather puzzling in the light of D, for if D is correct, it seems to show that a class of structures which are natural from one point of view (i.e. centre embeddings) are extremely unnatural from another. It is not necessarily to be expected that human linguistic abilities have evolved in a harmonious and homogeneous manner, but other things being equal, we would not expect to find two apparently co-operating modules so ill-suited to each other. Why should grammars be able to generate things that parsers can't parse?

When we consider left and right recursions, some further tension between A, B and D emerges. Multiple left recursions in English are most clearly illustrated by possessive determiner phrases, which are generally assumed to have a structure something like 3:

3



and multiple right recursions by a variety of structures, for example, relative clauses:

4 That's [the company that manufactures [the drugs that have [the side effects that made her come out in a rash]]]

There are several facts which suggest that the structures assigned to these examples by a grammar in accordance with A cannot realistically be assumed to play any very direct role in the actual processing of them in performance. Firstly, there is the familiar observation (Chomsky 1965: 13-14, Langendoen 1975: 544), that examples like 3 and 4 do not have the intonation contours that would be predicted for them on the basis of the constituent structures assigned to them by a grammar. For example, in 3, the intonation of the sequence of possessives is not defined over the whole constituent, as might be expected, but is more like a 'list' intonation. In sentences like 4, the intonation contour sometimes breaks the sentence up in the way indicated informally here:

5 [That's the company] [that manufactures the drugs] [that have the side effects] [that made her come out in a rash]

This chunking of the sentence does not respect its syntactic structure, splitting the head NP of the relative clause from its modifier and grouping it with the main clause instead. The conditions under which this happens are clearly connected with matters of length and so on, so the actual examples here are also capable of receiving the 'correct' contour, but the effect is clearly to be seen in longer and more complex sequences. This observation is generally taken to indicate that, whatever else is happening in the production and comprehension of such examples, it is not the case that complete syntactic structures of the type assigned by a grammar are being computed.

A further argument that this is so derives from the fact that although 4 was displayed as a right branching structure, it would also receive a left branching analysis, and if sufficiently complex, all possible combinations of the two. This means that the number of parses such a structure would receive goes up massively with the number of clauses involved (see Church and Patil 1982 for discussion of this. Analogous comments hold for PP modifiers and conjunctions on most analyses). It is clearly stretching credibility to assume that a parsing procedure follows very faithfully what a grammar says about such cases.

While difficult to reconcile with A (and hence B) these observations are consistent with D. This perhaps needs some elaboration: it is a reasonable conjecture, given what we know about short term linguistic memory, that the human parsing mechanism operates in a way which has the formal properties of a finite state device (see e.g. Chomsky 1963, Chomsky and Miller 1963, or, more recently, Langendoen and Langsam 1984). The fact that unlimited right or left recursion can be recognised, whereas centre recursion cannot, is consistent with this, for any (CF) language with bounded centre embedding is also a finite state language. However, when we turn to full parsing, as opposed to recognition, it turns out that the proper analysis even of left and right recursion demands non-finite-state resources (Langendoen 1975). Intuitively, this is easily seen: parsing a language can be regarded, abstractly, as a transduction from strings of terminal items to labelled bracketings representing structural descriptions. For the labelled bracketings to be well formed, left and right brackets bearing the same label must be paired up correctly. In the case of recursion, this means that the bracket language contains cases where some number of left brackets of type X must be paired up with the same number of right brackets of type X, for any number. This is a classic non-finite state language, and thus even if the input to the transducer is finite state, the overall transduction must be at least of context-free power, given no finite bound on recursion. Full parsing, therefore, of structures like 3 and 4, will demand resources of at least this power.

Let us now assume that D should be taken to apply, not just to cases of centre embedding, but to all types of recursion (as in Miller and Isard's original (1963) discussion of centre embedding). This is, in effect, a conjecture that the human parsing mechanism is forced to operate with no more than finite state resources, even though the class of languages generated by the grammars found natural by human beings might lie far outside the finite state class. Under such circumstances it would be expected that in the left and right recursive cases, a full parsing would not always be available, an expectation that we may take to be supported by the intonational evidence, and by the combinatorial explosion considerations alluded to above.

If this is a plausible line of reasoning, it nevertheless presents us with a further difficulty in the light of observation B. For if semantics is driven by syntax, it would seem to follow that structures which are not properly parsed should not be fully interpretable either. While this is clearly the case for centre embeddings, it is not the case for either left or right recursion: semantically speaking they are completely unproblematic. This is a further conflict which our model of parsing will have to resolve.

An Incremental Parser-Interpreter

My aim was to develop a parser and interpreter which was compatible with A to D, resolving the apparent conflicts between them, and which also incorporated in a fairly concrete form the assumption that grammars have some status, independently of parsers, as mental objects. That is to say, it was assumed that what linguists say about natural language in the form of a grammar (including semantic interpretation rules) is available to the parser-interpreter as some kind of data structure having roughly the form that the linguist's pencil and paper description would suggest. The aim was also to demonstrate a serious commitment to C by getting the parser to build up explicit representations of the meaning of a sentence piece by piece during the course of a parse. To my knowledge, the only other work which takes this commitment seriously at the appropriate level of formal detail (there is no shortage of well intentioned hand-waving) is that of Ades and Steedman (1982). In Pulman (forthcoming), I discuss some of the similarities and differences between these two approaches.

For purposes of illustration, I will assume that the underlying grammatical theory involved is some form of Phrase Structure Grammar, where semantic interpretation consists of translation into a simple form of higher order logic. Neither of these assumptions is crucial: the parsing procedure can be adapted to certain types of transformational grammar, and the associated process of semantic interpretation requires only that the semantic theory can be driven by syntactic structures, and that there is some way of doing function application and composition. It is unlikely that this this rules out any candidates at all.

The procedure is best thought of as a type of stack-based shift-reduce algorithm, though with the ability to deal with incomplete constituents. In the current implementation it operates non-deterministically: I (and others) have argued elsewhere (Pulman, forthcoming) that there is no good reason to suppose that parsing (as opposed to a more global process of comprehension) is deterministic. (Contra Marcus 1980, Berwick and Weinberg 1984. See also Crain and Steedman, forthcoming; Briscoe 1984).

The driving mechanism of the parser-interpreter maintains an agenda of configurations, each representing a particular state of a parse. A configuration is a pair consisting of a representation of the state of the stack, and the current position in the input string. The stack is a list of entries, of which (usually) only the top two are accessible to the basic operations of the parser. Each entry represents a wholly or partially recognised constituent, along with its interpretation in terms of a translation into a logical expression. An entry is a triple, consisting of a category label, indicating what type of constituent is being recognised, a 'needed' list of constituents which must be found before the category is complete, and the interpretation so far. The parser starts with an initial configuration and proceeds by trying to produce new ones from that until either no more alternatives are left, and the parse has failed, or one or more complete parses are produced.

There are four basic operations which produce a new configuration from an old one. Which one is performed depends on the state of the stack. If there is a choice between two, both are performed, producing two new configurations.

SHIFT: takes the next word from the input and creates a new stack entry for it (for each lexical entry it has in the dictionary). For example, given a lexical entry like

{every, Det, λ P λ Q Ax Px $\rightarrow$ Qx}

Shift produces a stack entry like:

{Det, nil, λ P λ Q Ax Px $\rightarrow$ Qx}

The interpretation of non-logical words is assumed to be the associated constant, as is customary. Since lexical categories are always complete the second 'needed' element in a stack entry will always be empty. Having created a new stack entry, Shift records a new configuration with that entry on top of the stack, and an updated input pointer.

INVOKE-RULE: applies when there is a completed entry on top of the stack. Essentially, it checks the rules in the grammar to see whether the category represented by that entry could begin some higher level constituent. Although this is not strictly necessary, a one-word lookahead is incorporated for efficiency.

If Invoke-rule succeeds in matching a category of an entry with the first member of the right hand side of a rule, it creates a new entry from them. Logically speaking, this process happens as follows: assume, for illustration, an entry of the form

{Det, nil, every}

(where the interpretation of 'every' might actually be as above) and a example rule of the form:

NP $\rightarrow$ Det N ; Det' (N')

where the part ofter the semi-colon is the semantic component. The entry matches the beginning of the right hand side of the rule and so could begin an NP constituent. Now assume a function, call it Abstract, which when applied to a rule of this form produces from its right hand side and semantic component the result of lambda abstracting over all the right hand side symbols (in the order specified in the rule) which appear in the semantic component. Thus Abstract applied to the rule above would produce

λ det λ n { det (n)}

If applied to a rule like

S $\rightarrow$ NP VP ; VP' (NP')

it would produce

λ np λ vp { vp (np)}

This is simply a more literal rendering of what the rule actually says, in fact: making explicit the fact that the items occurring in the semantic part of the rule are to be interpreted as variables.

When Invoke-rule has matched an entry to a rule it produces a new entry where the category is the left hand side of the rule, the 'needed' list is all but the first of the right hand side, and the interpretation is the result of applying Abstract to the rule and then applying that to the interpretation of the original entry. In the example above the result of all this would be:

{NP, N, λ n { every (n)} }

In other words, the interpretation is simply that of the whole rule with that of the existing entry put in the appropriate place: a semantic equivalent of the 'needed' field. In general, the interpretation of an incomplete constituent is that it is a function expecting to find the needed items as arguments.

COMBINE: combines a complete entry on top of the stack with an incomplete one below it, if the category label of the former matches the first 'needed' item of the latter. For example, if the stack contained an entry like the one just described, with a complete entry on top:

{N, nil, man}
{NP, N, λ n { every (n)} }

then Combine would produce a new entry with the category of the incomplete one, the remainder, if any, of the needed list, and an interpretation which is the result of applying that of the incomplete entry to that of the complete one. Here the result would be:

{NP, nil, every (man) }

when beta reduction of the lambda expressions has taken place, which is a complete constituent, in this instance, although this need not be the case. If the needed field is not nil, the interpretation will always reflect this.

These three operations are in fact sufficient to allow the parser to operate. However, a further operation is also necessary if we are to maintain consistency with our original assumptions.

CLEAR: Clear is intended to correspond to the intuition that once a complete or completable representation of a proposition has been built up, the syntactic information needed to do this is no longer required, under normal circumstances. The conditions under which Clear operates in the present implementation ensures that this type of syntactic information is discarded as soon as possible: although this is probably not a realistic claim about human parsing.

Clear operates when:

(i) there are only two items on the stack (in a less enthusiastic version, Clear would be constrained to operate only on the bottom two items on the stack)

(ii) the topmost one potentially contains everything needed to complete the bottom one

(iii) the topmost one is a VP or S

The first two conditions correspond to the obvious truth that you can only get rid of syntactic information when it is safe to do so, and that 'selective forgetting' is not possible: either all the syntactic information relevant to the earlier portion of the sentence is discarded, or none of it is. Otherwise, the claim, and the later explanations which depend on it, would be vacuous. The third is intended to capture the intuition that it is the main predicate of a sentence which when encountered provides enough information to be able to continue parsing safely after that point with no reference to anything before. For example, when a verb is encountered, the number and type of (obligatory) arguments will be known.

When the conditions for Clear are met, the effect is that the interpretation of the bottommost entry is composed with that of the topmost, the bottom one then being erased. For example, in a situation like:

{VP, NP, λ np {likes (np)} }
{S, VP, λ vp {vp (some (man))} }

where the topmost entry is of the type that the one underneath is looking for, the result of Clear is that the stack will contain just:

{VP, NP, λ x {λ vp {vp (some (man))} {λ np {likes (np)} (x)}}}

When this VP finds the NP it is looking for, the interpretation will reduce to what we would have had more directly if Clear had not operated.

Here is a trace of the parser to show how all these operations work together. The meanings of the individual lexical items have been suppressed in the interests of readability.

S → NP VP ; VP (NP)
VP → V NP ; V (NP)
NP → Det N ; Det (N)

Input: The farmer killed the duckling

Shift:
{Det, nil, the}
Invoke:
{NP, N, λ n {the (n)} }
Shift:
{N, nil, farmer}
{NP, N, λ n {the (n)} }
Combine:
{NP, N, the (farmer) }
Invoke:
{S, VP, λ vp { vp (the (farmer))} }
Shift:
{V, nil, killed}
{S, VP, λ vp { vp (the (farmer))} }
Invoke:
{VP, NP, λ np { killed (np)} }
{S, VP, λ vp { vp (the (farmer))} }
Clear:
{VP, NP, λ x {λ vp {vp (the (farmer))} {λ np {killed (np)} (x) }} }
Shift:
{Det, nil, the}
{VP, NP, λ x {λ vp {vp (the (farmer))} {λ np {killed (np)} (x) }} }
Invoke:
{NP, N, λ n {the (n)} }
{VP, NP, λ x {λ vp {vp (the (farmer))} {λ np {killed (np)} (x) }} }
Shift:
{N, nil, duckling}
{NP, N, λ n {the (n)} }
{VP, NP, λ x {λ vp {vp (the (farmer))} {λ np {killed (np)} (x) }} }
Combine:
{NP, nil, the (duckling)}
{VP, NP, λ x {λ vp {vp (the (farmer))} {λ np {killed (np)} (x) }} }
Combine: {VP, nil, λ x {λ vp {vp (the (farmer))} {λ np {killed (np)} (x) }} (the (duckling)) }

At this point the parse is complete, and the complex interpretation beta-reduces to:

{killed (the (duckling))} (the (farmer))

The resulting interpretation is exactly what would have been obtained by a 'classical' system operating as described earlier.

Modelling Incremental Interpretation

How does the parsing procedure manage to remain faithful to A to D simultaneously? Let us begin with B: the compositional, syntax-driven nature of semantics. The parser assumes that semantic information can be associated with syntactic rules in some way (though it is not ruled out - in fact, it is assumed - that some extra aspects of interpretation may need to be computed by separate procedures: for example, identification of variables for the purposes of indicating coreference; cases of wide scope of quantifier phrases in syntactic narrow scope positions, etc.). Once the rule in question has been identified by Invoke-rule, the semantic information involved is extracted and used to form the next stack entry. The syntactic information is also used to form expectations about what constituents must come next, although it is conceivable that if semantic type is entirely predictable from syntactic category and vice versa this information is actually redundant. No other mechanisms for linking syntax with semantics are required. Hence the parser obeys condition B absolutely literally and faithfully.

The important thing to notice is that this is achieved without building any explicit syntax trees during the course of parsing a sentence. Syntactic information is used to build up the interpretation and to guide the parse, but does not result in the construction of an independent level of representation. As the title of the paper indicates, there is no parse tree built for a sentence at all. While it is true that in some sense trees are implicit in the sequence of operations of the parser, this is an inevitable consequence of the fact that the rules used themselves define trees, and as we shall see, even in this weak sense the tree structures implicit for certain types of recursive construction are not isomorphic to those which would be defined by the grammar.

I like to think of this aspect of the operation of the parser as embodying the intuition often expressed (most often in the oral tradition, though explicit in Isard 1974), that syntax is a 'control structure' for semantics. It also has the merit of being consistent both with the widespread agreement among linguists that syntax plays a central role in language understanding, and with the apparently equally widespread failure of psycholinguists to find any evidence that purely syntactic representations are computed at any stage during normal comprehension.

Turning now to C, the observation that sentences are understood on (at least) a word by word basis on a pass through from left to right, it should be clear that our procedure provides a direct model of this process, on the assumption that at least a central part of the meaning of a sentence is given by a translation into a logical form of this kind. As soon as a word is encountered, it is integrated into the logical form being built up. At every stage, this logical form, though possibly not yet complete, is a perfectly meaningful object (within the higher order logic assumed here it is just a term like any other): it can be used to perform inferences, be the antecedent for anaphora or ellipsis, be integrated with the context so as to assess and discard alternative interpretations corresponding to different parsings, and in general perform any of the functions we expect the meaning of a sentence or sentence fragment to be able to do.

The satisfying of A is in a sense automatic but trivial, given that the parser uses ordinary grammatical rules, rather than some preprocessed version altering the output of the rules to produce flat structures (as, for example, in Langendoen 1975, Langendoen and Langsam 1984, and also - wrongly, on the present approach - in Pulman 1983). More interesting is the way the parser produces a similar effect to that achieved with these preprocessings, without altering the rules themselves, as a side effect of its observance of D - the limitation on recursion.

Recursion Limitations

I have argued elsewhere (Pulman, forthcoming) that attempts to explain the difficulty of centre embedded sentences as a consequence of parsing strategies are unsuccessful, and that the simplest explanation is the original one (Miller and Isard 1963): that the human parsing mechanism is fundamentally incapable of operating recursively. To be more precise: if (in the worst case) the parser encounters an instance of a construction in the course of trying to parse an earlier instance of it, the record of the earlier instance will be erased and 'forgotten', causing confusion in those cases where the information is needed to complete a parse successfully, as in the centre embedding cases. Clearly this is not absolute: some instances of centre embedding can be found to a depth of 4 or 5, but for simplicity we will assume that there is some small fixed limit, L.

The present procedure implements this restriction quite literally: if Invoke-rule attempts to put on the stack an incomplete constituent of category X, when there are already L instances of such incomplete Xs on the stack, then the earliest instance is erased before Invoke-rule can succeed. The interesting and striking thing about this restriction is that as stated, it applies to all types of recursion, and thus might be expected to result in parsing failures not just for centre embedded examples of a depth greater than L, but for left and right recursions deeper than L too. However, this does not happen: the basic operations of the parser in fact conspire to bring it about that both left and right recursions can be parsed, the former fully, and the latter to just the extent, apparently, that is needed to be able to provide them with an appropriate interpretation. Thus a perfectly general and simple restriction can be imposed, rather than some version (implausibly) qualified so as to distinguish between different types of recursion.

The simplest case is that of left recursion, which we will illustrate with an artifical example grammar:

A → Aa ; A (a)
A → a ; a

When processing a string 'aaa...', the parser operates as in the following trace ('b' is the interpretation of 'a'):

Shift:
{a, nil, b}
Invoke:
{A, nil, b}
Invoke: {A, a, λa {b (a)}}
Shift:
{a, nil, b}
{A, a, λa {b (a)}}
Combine:
{A, nil, b (b) }
Invoke:
{A, a, λa {b (b (a))} }

At this point the cycle of operations has become evident: at no point is there ever more than one occurrence of an incomplete A constituent on the stack, and so there is never any situation in which the recursion limitation would come into effect. In other words, like any shift-reduce mechanism, this parser can process unbounded left recursion without the stack growing beyond a constant depth.

Centre embeddings of a depth greater than L will not be parsed correctly. To see how this might work out in detail we will assume some simple rules for relatives:

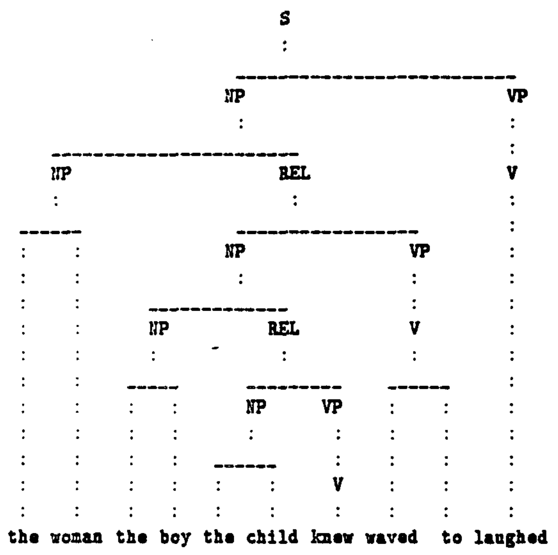
NP → NP REL : REL'(NP')
REL → NP VP : NP'(VP')

and we will ignore the question of how wh words are linked with gaps appropriately, other than the assumption that this information is contained somewhere in the trees defined by these rules. Notice that we are assuming for simplicity that relative clauses are a distinct constituent from S, and also, that no recursion at all is allowed. For clarity, rather than build the incremental semantic interpretations yielded by the parser we will display the partial tree that a more conventional parser might build.

For a sentence like:

7 The woman the boy the child knew waved to laughed

we ought to build a tree like:



Things proceed as follows, ignoring some obvious steps:

(i) { NP, nil, {NP the woman}}

(ii) { NP, REL, {NP {NP the woman}{REL ...}}}

(iii) {NP, nil, {NP the boy}}
{ NP, REL, {NP {NP the woman}{REL ...}}}

At this point, if we are to find the correct interpretation or build the appropriate parse tree Invoke must recognise the NP 'the girl' as the beginning of another relative clause, and place on the stack an entry like:

{NP, REL, {NP{NP the girl}{REL ...}}}

But of course this will violate the recursion restriction, for there is already an {NP, REL...} on the stack. Let as assume that this earlier one is thus 'forgotten', or at least rendered inaccessible to the parsing procedure in some way. Things now proceed - again ignoring obvious details - until we have recognised the sentence as far as the word 'knew':

(iv) {NP, nil, {NP {NP the girl}{REL {NP the boy}{VP knew}}}}

At this point the procedure runs into trouble. If the parser merely continues with 'waved to' it will be stuck: 'waved to' in its own is not a complete VP, for it is missing an object. So a possible parse in which what is on the stack is the subject of 'waved to' will fail. But there is no other option available for it. In order to treat 'waved to' correctly, the parser needs to know that it is part of a relative clause and thus can legitimately have a missing object. But this of course is precisely the information that is no longer available to it, for the REL entry which would have signalled this has been erased. So the parser cannot proceed beyond this point coherently. It is reassuring that this is exactly the point - after the first verb of the sequence stacked up - where both intuitive and experimental evidence (Miller and Isard 1963) suggest the onset of difficulty with these constructions. Our parsing procedure seems to get stuck at exactly the same point people do in these centre embedded constructions.

With right recursion there are two cases of interest. With multiple sentential complementation like

9 Joe thought that Bill expected that Mary knew ....

then the operation of Clear means that the recursion limit will never be exceeded. Whenever we have a stack of the form:

{VP, S, beta}
{S, VP, alpha }

Clear will erase the bottom entry leaving:

{VP, S, λx {alpha {beta (x)}} }

Whenever there is a stack of the form:

{S, VP, beta}
{VP, S, alpha}

Clear will likewise produce:

{S, VP, λx {alpha {beta (x)}} }

Thus neither recursive category will ever have more than one instance on the stack at a time. As in the earlier illustrative examples, the process of function composition means that, when the final constituent is encountered, the whole complex logical expression reduces down to exactly what we would have had under the 'classical' view: the difference here is that we do not depend on the whole syntactic tree being explicitly constructed first in order to get the correct results.

While the general idea here seems correct, the details are not entirely satisfactory, however. In the current implementation, Clear operates whenever it can, which, as remarked above, does not seem very plausible. Since the motivation for Clear is partly via considerations of short term memory load, in a more realistic model some extra parameter to reflect this transient load should clearly be involved, such that Clear only operates when a certain threshold is exceeded. This would mean that there was room for some decoupling of the recursion limitation from the conditions on Clear: at present, with a recursion limit of 1, even a sentence like

10 John expected that Bill would leave

could not be parsed unless Clear had operated. But it seems unlikely that such a short sentence imposes any very great strain on syntactic short term memory. Furthermore, in the present implementation, Clear will prevent sentential conjunctions from being parsed at all, for by the time the conjunction is reached, the only constituent left on the stack is labelled as a VP, not an S, and so Invoke-rule cannot find an appropriate candidate to continue. Fortunately, both of these wrinkles are easily amended by making Clear more conservative in its operation, while preserving the present type of explanation for why this type of right recursive construction can still be parsed with little apparent effort.

Not all cases of right recursion need be 'rescued' by Clear, however. Given multiple PP modifiers, introduced by a rule:

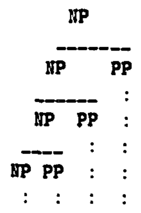
NP $\rightarrow$ NP PP

we have the potential for the type of situation described earlier, where there may be many distinct parse trees, only one of which may accurately reflect the actual pattern of attachment of PPs to the NP they modify.

11

The house in the woods by the river
The book on rock climbing by the writer from Scotland
The bird in the tree near the flowerbed with a red beak

Assuming a recursion limit of 1, there is only one 'parse' of such structures that will succeed, since Clear - applying only to projections of +V, recall - cannot be involved. The parsing procedure will process these cases in a way which corresponds to a left branching or stacked analysis:



This might seem to be a serious disadvantage, for there are clearly readings of the above examples which appear not to be those suggested by such a 'parse'. However, it is actually a good result: when there is more than one parse of a sequence like this, the 'correct' one - i.e. that consistent with the preferred attachments - must be decided on by a mixture of semantic and contextual constraints on what can modify what. A full and exhaustive parse is thus still not sufficient to arrive at a unique interpretation. But if the real work of deciding what attachments are to be made is done by these non-syntactic procedures, then all but the lowest level of syntactic analysis, (into non-recursive NP and PP constituents), is entirely redundant. All but one of the more complex analyses will be thrown away, and all of the semantic information to be gained from that analysis has already been computed in the course of deciding that it is the 'correct' one. (As everyone who has ever written a practical parser has discovered, this is in any case an extremely silly way to do things). Thus an exhaustive syntactic analysis is neither necessary nor sufficient for the correct handling of these sequences. All that is required is that the low level constituent structure be recognised: thereafter, the meaning of a modifier can be assumed to be a function which seeks an appropriate argument to modify, and is thus just applied to the representation of the meaning of the sentence that has already been built up. Incidentally, notice that this latter assumption is almost forced on us independently by the existence of rightward extraposed nominal modifiers which may be encountered without warning after an apparently complete sentence meaning has been assembled:

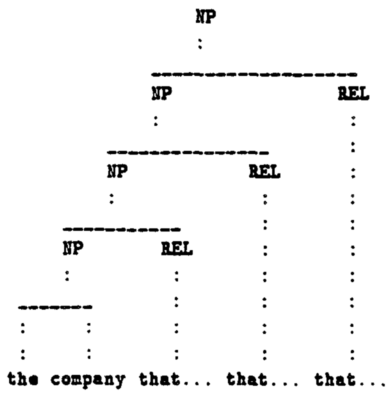
12 I gave the book back to the girl in the library that you asked me to photocopy

The level of analysis provided by our treatment appears to be exactly what is needed for the attachment of these modifiers to be accommodated appropriately.

Sequences of ordinary relative clauses, and multiple conjunctions will be treated in a similar way, and similar arguments apply to them. In the case of conjunctions, of course, the fact that no information is lost by not computing massive parse trees is even more obvious.

It is interesting to note, in connection with sequences of relatives, that the stacked 'parse' which the operation of the procedure mimics is actually the one which corresponds almost exactly to the unexpected intonation patterns noted by Chomsky and Langendoen:

13 {That's the company} {that manufactures the drugs} {that have the side effects} {that made her come out in a rash}



In general, then, the recursion limitation and the basic operations of the parser-interpreter seem to combine to provide a fairly satisfactory model of the parsing and understanding of these different types of recursive constructions.

Summary

I have presented an algorithm for parsing and interpreting grammars and semantic descriptions of a certain formal type, which is consistent with a set of clear and uncontroversial facts about human linguistic performance. In particular, I hope to have show that a (partial) theory of competence can be literally embedded within a model of performance, in such a way that simple principles belonging to the latter (recursion limitations) explain phenomena that have sometimes been taken to pertain to the former.

There are some further practical consequences arising from this work: there is not space to go into the details here, but there is an interpretation of the parsing algorithm above - as one might suspect, given its formal properties - as a finite state transducer mapping strings of (labelled) terminal items directly into logical forms. While the construction of such a device from a grammar of the original type is rather complex, the result would be a 'linguistic engine' (sentences in, logical forms out) of formidable efficency.

Footnote

The parser-interpreter is written in Franz Lisp under 4.2 Unix on a Sun workstation. The current grammar provides syntactic and semantic coverage for simple complement types, phrasal and sentential conjunction, relative clauses, and questions.

References

Ades, A. E. and Steedman, M. J. (1982)

On the Order of Words. Linguistics and Philosophy 4, 517-558

Berwick, R. C. and Weinberg, A. S. (1984)

The Grammatical Basis of Linguistic Performance: Language Use and Acquisition. Cambridge, Mass: MIT Press.

Briscoe, E. J. (1984)

Towards an Understanding of Spoken Speech Comprehension: the Interactive Determinism Hypothesis. Ph. D. Diss., Dept. of Linguistics, Univ. of Cambridge.

Chomsky, N. (1963)

Formal Properties of Grammars. In R. Luce, R. Bush and E. Galanter (eds) Handbook of Mathematical Psychology Vol II, New York: John Wiley.

Chomsky, N. (1965)

Aspects of the Theory of Syntax. Cambridge, Mass: MIT Press.

Chomsky, N. and Miller, G. (1963)

Finitary Models of Language Users. In R. Luce, R. Bush and E. Galanter (eds) Handbook of Mathematical Psychology Vol II, New York: John Wiley.

Church, K. W. and Patil, R. (1982)

Coping with Syntactic Ambiguity, American Journal of Computational Linguistics, 8, 139-149

Crain, S. and Steedman, M. J. (forthcoming)

On Not Being Led Up The Garden Path: The Use of Context by the Psychological Parser. In A. Zwicky, L. Kartunnen, and D. Dowty (eds) Natural Language Parsing: Psycholinguistic, Theoretical, and Computational Perspectives, Cambridge: Cambridge Univ. Press.

Gawron, J. M. et al (1982)

The GPSG Linguistics System, Proceedings of the 20th annual meeting, Association for Computational Linguistics.

Isard, S. (1974)

What would you have done if...? Theoretical Linguistics, Vol 1, 233-256

Katz, J. J. (1981)

Language and Other Abstract Objects Oxford: Basil Blackwell.

Langendoen, D. T. (1975)

Finite State Parsing of Phrase Structure Languages and the Status of Readjustment Rules in Grammar. Linguistic Inquiry 6, 533-554.

Langendoen, D. T. and Langsam, Y. (1984)

The Representation of Constituent Structures for Finite State Parsing in Proceedings of Coling 84, Association for Computational Linguistics.

Miller, G. and Isard, S. D. (1964)

Free Recall of Self Embedded English Sentences. Information and Control 7, 292-303.

Pulman, S. G. (1983)

Generalised Phrase Structure Grammar, Earley's Algorithm, and the Minimisation of Recursion in Sparck Jones and Wilks eds.

Pulman, S. G. (forthcoming)

Computational Models of Parsing in A. Ellis (ed) Progress in the Psychology of Language, Vol 2, Lawrence Erlbaum Associates Ltd.

Rosenschein, S. J. and Shieber, S. M. (1982)

Translating English into Logical Form Proceedings of the 20th annual meeting, Asssociation for Computational Linguistics.

Schubert, L. K. and Pelletier, F. J. (1982)

From English to Logic: Context Free Computation of 'Conventional' Logical Translation, American Journal of Computational Linguistics, 8, 27-44.

Sparck Jones, K., and Wilks, Y. (eds) (1983)

Automatic Natural Language Parsing, Chichester: Ellis Horwood Ltd.