A DETERMINISTIC PARSER WITH BROAD COVERAGE

Robert C. Berwick

MIT Artificial Intelligence Laboratory, Room 820
545 Technology Sq., Cambridge, MA 02139

ABSTRACT

This paper is a progress report on a series of three significant extensions to the original parsing design of (Marcus 1980).* The extensions are: The range of syntactic phenomena handled has been enlarged, encompassing sentences with Verb Phrase deletion, gapping, and rightward movement, and an additional output representation of anaphor-antecedent relationships has been added (including pronoun and quantifier interpretation). A complete analysis of the parsing design has been carried out, clarifying the parser's relationship to the extended LR(k,t) parsing method as originally defined by (Knuth 1965) and explored by (Szymanski and Williams 1976). The formal model has led directly to the design of a "stripped down" parser that uses standard LR(k) technology and to results about the class of languages that can be handled by Marcus-style parsers (briefly, the class of languages is defined by those that can be handled by a deterministic, two-stack push-down automaton with severe restrictions on the transfer of material between the two stacks, and includes some strictly context-sensitive languages).

I EXTENDING THE MARCUS PARSER

While the Marcus parser handled a wide range of everyday syntactic constructions, there are many common English sentences that it could not analyze. One gap in its abilities arises because it did not have a way to represent the possibility of rightward movement -- that is, cases where a constituent is displaced to the right:

A book [about nuclear disarmament] appeared yesterday. -->
A book appeared yesterday [about nuclear disarmament].

Further, the only way that the Marcus parser could handle leftward movement was via the device of linking a "dummy variable" (a trace) to an antecedent occurring somewhere earlier in the sentence. For instance, the sentence, "Who did Mary kiss?" is parsed as, **Who did Mary kiss** *trace*?, where *trace* is a variable bound to its "value" of *who*, indicating the intuitive meaning of the sentence, "For which X, did Mary kiss X".

In the original parser design, a *trace* was of the category NP, so that only Noun Phrases could be linked to traces. But this meant that sentences where other than NPs are displaced or deleted cannot be analyzed. This includes the following kinds of sentences, where deleted material is indicated in square brackets.

Gapping:
Max gave Sally a nickel, and Harvey [gave Sally] a dime.

VP deletion:

John kissed Mary, and I think that Frank said that Mary thought that Harry would have [kissed Mary] too.

The new parser is also designed so as to account for the ill-formedness of certain sentences related to these, such as the "gapped" sentence,

John hit Mary and I don't believe Bill [hit] Sue.

The new design actually explains *why* gapping is bounded in a way that VP deletion is not. Finally, since languages such as German arguably contain rules that move Verbs (the German "Verb Second" rule; see (Thiersch 1978), an extension to the trace system is demanded here as well.

The last class of new cases to be handled includes the encoding of information not dealt with in the original Marcus parser, that of overt antecedent-anaphor binding. Such sentences include:

Reciprocals:
They think that feeding each other would be dangerous.
{They = each other}

Pronouns:
John thinks that he is wonderful.
{John = he, under one interpretation}
His mother likes John.

How can we begin to attack these cases? Let us consider rightward movement first. It is a simple descriptive fact about such sentences that the moved constituent has been displaced from its "normal" location. By analogy with the case of leftward movements, the obvious first tack is to place a variable in the position from which the constituent has been moved, and then bind that variable to the appropriate constituent appearing to its right:

A book [t] appeared yesterday about nuclear disarmament.

This solution is not quite correct, however, since it fails to explain a key property about rightward movement (in English): unlike leftward movement, it is restricted to the domain of a single S(entence). But if right- and leftward movements were determined by the same mechanism, there should be no such asymmetry, all other things being equal. However, there is solution within the framework of the deterministic Marcus parser that accounts for this asymmetry. Since the constituents "a book" and "a book about nuclear disarmament" are both completely well formed, there is no way for a deterministic parser to be able to record the possibility that "a book" may be lacking a complement ("about nuclear disarmament") unless there is explicit evidence in the input stream within a bounded distance of the NP "a book". This is because all rules in the parser must be stated in a finite "IF-THEN" format that can make reference to at most three constituents of look-ahead information, a bound that prohibits one from moving "about nuclear disarmament" too far to the right. The idea, then, is that instead of adding a variable [t] to the NP "a book", we simply flag

* This paper reports work done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is supported in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Reseaarch contract N0014-80-C-0505.

the NP with the diacritic "+rightmoved" *if there is explicit evidence in the look-ahead buffer that this is correct.* This flag indicates that the argument structure of the NP is not yet complete (it lacks a complement). Observe that in the case above the lookahead buffer can contain a proper triggering pattern, since at the time "a book" must be marked, the first cell in the buffer will contain "appeared", the second, "yesterday", and the third, "about". (More generally, a complete NP as discussed in (Marcus 1980) could reside in the second lookahead cell, and the a PP in the third. Note that the type of the PP will determine whether it is a possible complement of the Subject NP.) Then, when the PP complement is encountered and parsed, it is literally attached to the NP that is so flagged. (Note that this is in contrast to the "trace" solution, where variable binding is used to effect the "restoration" of initial argument structure. Here, the NP is "reconstructed".)

What of sentences with deleted Verb Phrases or "gapped" constituents? Again, the key distinction is that "gapping" is bounded in a way that VP deletion is not (as illustrated by the VP deletion sentence above where the deleted VP "kissed Mary" can be arbitrarily far from its copy). A "gapped" constituent, in contrast, is locally detectable, and so we can attempt to formulate local IF-THEN grammar rules to handle it. Importantly, there appear to be constraints on natural grammars that prohibit identical gapped constructions that could have been derived from two different underlying sources -- just what is needed in order to accommodate these examples in a deterministic parser. Further, it seems that gapped constituents must lie at the left or right periphery of complete sub-trees. For a discussion of these constraints, see (Hankamer 1973). Lack of space prevents a complete description of this rule system here, but to take just one example, consider the following sentence (along with its underlying form):

Max gave Sally a nickel yesterday, and [Max gave Sally] a dime today.

The parser will analyze the first part of the conjunct, up to the "and", as a complete S. Next, "and" will be pushed onto the parser's stack, delaying the decision as to what to do until we look at material in the input buffer. At this point, the lexical items "a dime" in the input buffer triggers the analysis of an NP (note that "today" unambiguously marks the end of the NP). Following the Marcus parser design, this NP is returned to the input buffer, which now contains NP in its first cell, "today" in its second, and an end of sentence marker in its third cell. (Recall that in general the buffer is limited to contain just three items.) But the coordination structure "S and S" demands that a full S (NP VP) be found. The periphery constraint, in turn, requires that the deleted material form lie at the left or the right of "NP today", forming a complete subtree. This means that the only possible choices are [...] NP today or NP today [...] where [...] together with "a dime today" forms an S. But since the sequence NP today [...] is known not to form an S, the only remaining choice is [...] NP today. That is, the parser inserts a [e] constituent into the first position of the buffer and takes [e] NP today as forming an S. The interpretation of the [e] is left for a second stage of processing.

It is interesting to contrast this example with comparable VP deletion sentences. The deleted VP can be arbitrarily far away from its copy, but not the gapped constituent:

??Max gave Sally a nickel yesterday, and John thought that Mary believed a dime today.

Max gave Sally a nickel yesterday, and John thought that Mary believed that Bill did too.

The point is that by abiding by observable linguistic constraints, we can actually design a parser to efficiently handle the sentences of a natural language. (Conversely, the demand that sentences be efficiently parsable at least suggests a "functional" explanation for these kinds of constraints and evidence for deterministic parsing. However, this possibility must remain a suggestion, since the constraints are not necessary for efficient parsability.)

Observable differences in behavior indicate then that a parser should use different techniques to handle gapping and VP deletion. VP deletion cases are always unambiguous, in the sense that it is always clear when a VP is to be inserted into a sentence -- every (English) sentence requires a VP. For instance, this means that given the sentence,

John kissed Mary, and I think that Frank said that Mary thought that Harry would have too.

a VP may be inserted by the parser after the words "would have" because, using its lookahead buffer, the parser can see that there is no VP present (as indicated by the triggering word "too"). Thus one can write an IF-THEN grammar rule based on just local context to handle the surface parsing of these cases. It is precisely this property that permits VP deletion to be unbounded. (In contrast, the parse of a "gapped" construction demands that one examine the left-context of the parse to determine how to proceed.) This grammar rule inserts a VP into the sentence after "would have", without specifying any relationship between that VP and any previous VP. The actual *interpretation* of the deleted VP is carried out in a second stage of analysis, described immediately below. (The approach follows that described by (Williams 1977).)

Finally, the new parser is designed to compute a representation of general anaphoric relationships. It does this by implementing a restricted version of *co-indexing*, adding a pointer from an anaphor to a representation of an antecedent NP. The implementation is a restricted one because the pointer is directed: if there is a link from X to Y, then X is <u>dependent</u> on Y, in the sense of referential dependency. For example, given the sentence,

John thought that he liked Nixon.

One desired output representation (in co-indexed format) is:

John-1 thought that he-1 liked Nixon.

In its dependency implementation, the notation "he-1" means simply that a pointer is set up linking "he" to the representation of the NP "John". "He" is dependent on "John" in the sense that any properties attributed of "John" are automatically inherited by "he," but not necessarily vice versa. (For discussion of some of the advantages of a "dependency" representation over the more conventional scheme of simply setting the "values" of the NP antecedent and its anaphor to be equal, see (Higgenbotham 1982).) Observe that dependency links give only *possible* connections between items, not necessary connections; "he" can always be dependent on some discourse NP not mentioned in the sentence.

The constraints that dependency "linking" obeys have been explored in depth in recent years in syntactic theory and will not be covered here. Briefly, an antecedent can be linked to a pronoun if the pronoun is "free" in a certain local constituent domain. So for example, "he" is free in the domain [that he liked Nixon], and hence can be dependent on "John" outside this domain. In contrast, "him" and "Bill" must be distinct (i.e., "him" cannot depend on "Bill") in "Bill knows him". Note that the constraint cannot be that a pronoun must appear after an antecedent, since we can have "him" and "John" dependent in,

The guy who knows him likes John.

The local constituent domain definition works here though, since "he" is free in "who knows him" and thus is available for dependency linking to "John".

How are potential dependents calculated? Briefly, the procedure is to maintain an unordered set of previously

encountered NPs, partitioned according to the agreement features of person, number, and gender. The set is updated as local constituent domains are constructed by the (syntactic) parser. Potential antecedence is encoded by adding a pointer from the pronoun to the set, after checking for agreement. For example, consider the sentence:

John gave the bill to him, and then he left.

The first "local domain" relevant to pronoun linking is the sentence, "John gave the bill to him". The list of antecedent NPs is empty. On encountering "him", the program links "him" to it. (No compatibility violations are found since the set is empty.) Thus "him" has an empty antecedent with respect to this sentence, the correct result. If previous sentences has established discourse NPs, then "him" could be linked to a set of these. Next, on encountering "and then" in its input buffer, the parser notes that the local domain (the S) has been completely built. Therefore, the two NPs "John" and "him" are added to the appropriate candidate antecedent set. "John" and "he" remain in the same partition, since they agree in terms of defined features. Finally, during the analysis of the sentence, "he left" the parser again links a pronoun, "he" to the only antecedent set, correctly encoding the possibility that either "he"=John or "he"="him" (= some unspecified discourse NP). Note that these alternatives are left implicit in the set description, thus avoiding the computational complexity of "writing out" all the possibilities. Given this approach, it can be shown that the full syntactic analysis and linking algorithm runs in polynomial time (see (Berwick and Weinberg, 1983, forthcoming).)

Finally, in passing it should be noted that the revised parsing machinery permits one to extend the parser to handle other languages, e.g., German, using the "Move verb" analysis proposed by (Thiersch 1978). The details of this work are reported in (Lester 1982).

II FORMAL CHARACTERIZATION

The Marcus parser incorporated a number of interesting design features: the use of a lookahead buffer based on constituents (whole phrases) and not just words; the interweaving of top-down prediction of syntactic categories (as in an ATN) with bottom-up recognition; the use of "attention shifting" to delay the immediate analysis of some constituents; and pattern-action grammar rules grouped into packets. However, its informal characterization (as a set of grammar rules plus interpreter) has made it difficult to see just what class of languages could be handled by such a machine and what its computational complexity is. It is possible to take these program design features and formally model them, reducing the machine to a more well-known class of devices. To summarize this reduction here: (1) constituent lookahead follows precisely the notion of LR(k,t) parsing, as defined by (Knuth 1965) where t=the number of left-most complete subtrees that can be used in the lookahead. (2) The use of top-down prediction in such a parser does not change its basic bottom-up completion of phrases, as established by (Hammer 1974). This means that top-down prediction does not really affect the class of languages that this type of parser can handle. (3) LR(k,t) parsers can be modeled as two-stack automata, i.e., an input buffer and push-down stack, with constituents moved between input buffer and stack and vice-versa. If the input buffer can be of unlimited size, such a machine can be shown to be able to handle some strictly context-sensitive languages (in time n^2 where n=the length of input sentences). (4) In this framework, "attention-shifting" corresponds exactly to shifting an item onto the push-down stack from the input buffer, without reducing it to a higher category nonterminal symbol. Thus we can determine what class of languages Marcus-style parsers can handle and the computational complexity of these devices. (5) Rule packets record part of the left-hand context of the parse, corresponding to the item sets of an LR(k) (hence also LR(k,t)) parser. All the ingredients of Marcus-style parsers, then, have been formally studied in the context of programming language parsing, suggesting that known techniques for reducing the size of LR(k) parsers (see, e.g., Anderson, Eve, and (Horning 1973) may prove applicable to "compiled" versions of Marcus-style parsers. This possibility is currently under investigation.

REFERENCES

[1] Anderson, T., Eve, J. and Horning, J. "Efficient LR(1) Parsers," Acta Informatica, 2:1, (1973) 12-39.

[2] Berwick, R. Locality Principles and the Acquisition of Syntactic Knowledge, PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1982

[3] Berwick, R. and Weinberg, A. The Grammatical Basis of Linguistic Performance, Cambridge, MA: MIT Press, 1983 in press.

[4] Hammer, M. A New Transformation Into Deterministic Top-down Form, PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1974.

[5] Hankamer, J. "Unacceptable Ambiguity," Linguistic Inquiry, 4:1, (1973) 17-68.

[6] Higgenbotham, J. "Linking and Logical Form," unpublished ms., MIT Department of Linguistics and Philosophy, 1982.

[7] Knuth, D. "On the Translation of Languages From Left to Right" Information and Control, 8:6, (1965) 607-639.

[8] Lester, A. A Parser for Simple German Sentences, MIT Bachelor's thesis, Department of Electrical Engineering and Computer Science, 1982.

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

[10] Szymanski, T. and Williams, "Noncanonical Bottom-up Parsing," SIAM Journal of Computing, 1:2, (1976) 146-160.

[11] Thiersch, C. Topics in German Syntax, PhD thesis, MIT Department of Linguistics and Philosophy, 1978.

[12] Williams, E. "Discourse and Logical Form," Linguistic Inquiry, (1977) 101-140.