RESTRICTING LOGIC GRAMMARS

EDWARD P. STABLER, JR.

Quintus Computer Systems
2345 Yale St.,
Palo Alto, CA, 94306
stabler@quintus.uucp

ABSTRACT

A parser formalism for natural languages that is so restricted as to rule out the definition of linguistic structures that do not occur in any natural language can make the task of grammar construction easier, whether it is done manually (by a programmer) or automatically (by a grammar induction system). A restrictive grammar formalism for logic programming languages is presented that imposes some of the constraints suggested by recent Chomskian linguistic theory. In spite of these restrictions, this formalism allows for relatively elegant characterizations of natural languages that can be translated into efficient prolog parsers.

I. INTRODUCTION

The best-known parser formalisms for logic programming systems have typically aimed to be expressive and efficient rather than restrictive. It is no surprise that in these systems a grammar writer can define linguistic structures which do not occur in any natural language. These "unnatural" structures might suffice for some particular processing of some particular fragment of a natural language, but there is a good chance that they will later need revision if the grammar needs to be extended to cover more of the natural language. On the other hand, if the grammar writer's options could be limited in the right way, there would be less to consider when a choice had to be made among various ways to extend the current grammar with the aim of choosing an extension that will not later need revision. Thus a restricted formalism can actually make it easier to build large, correct, and upward-compatible natural language grammars. A similar point obviously holds for automatic "language learning" systems. If a large class of languages must be considered, this can increase the difficulty of the problem of correctly identifying an arbitrary language in the class. So there are certainly significant practical advantages to formalisms for natural language parsers which allow the needed linguistic structures to be defined gracefully while making it impossible to define structures that never occur.

Recent work in linguistic theory provides some indications about how we can limit the expressive power of a grammar notation without excluding any human languages. There appear to be severe constraints on the possible phrase structures and on the possible "movement" and "binding" relationships that can occur. The exact nature of these constraints is somewhat controversial. This paper will not delve into this controversy, but will just show how some of the constraints proposed recently by Chomsky and others, constraints to which all human languages are thought to conform, can very easily be enforced in a parsing system that allows an elegant grammar notation. These grammars will be called "restricted logic grammars" (RLGs). Two well known logic grammar formalisms, definite clause grammars (DCGs) and extraposition grammars (XGs) will be briefly reviewed, and then RLGs will be introduced by showing how they differ from XGs. RLGs have a new type of rule ("switch rules") that is of particular value in the definition of natural languages, and the automatic enforcement of some of Chomsky's constraints makes RLG movement rules simpler than XGs'. We follow the work of (Marcus, 1980), (Berwick, 1982) and others in pursuing this strategy of restricting the grammar formalism by enforcing Chomsky's constraints, but we use a simple nondeterministic top-down backtracking parsing method. This approach to parsing, which has been developed in logic programming systems by (Pereira and Warren, 1980) and others, allows our rules to be very simple and intuitive. Since, on this approach, determinism is not demanded, we avoid Marcus's requirement that all ambiguity be resolved in the course of a parse.

II. DEFINITE CLAUSE GRAMMARS (DCGs)

DCGs are well known to logic programmers. (See Pereira and Warren, 1980 for a full account.) DCGs are similar to standard context free grammars (CFGs), but they are augmented with certain special features. These grammars are compiled into prolog clauses which (in their most straightforward use) define a top-down, backtracking recognizer or parser in prolog.

A DCG rule that expands a nonterminal into a sequence of nonterminals is very similar to the standard CFG notation, except that when the right-hand side of a rule contains more than one element, some operator (like a comma) is required to collect them together into a single term. The rules of the following grammar provide a simple example:

```
s --> np , vp.      det --> [the].
np --> det , n.     n --> [woman].
vp --> v.           v --> [reads].
```

(DCG 1)

The elements of the terminal vocabulary are distinguished by being enclosed in square brackets. An empty expansion of a category "cat" is written "cat --> []." (DCG 1) defines a simple context free language which includes "the woman reads".

Two additional features provide DCGs with considerably more power. First, the nonterminals in the DCG rules may themselves have arguments to hold structural representations or special features, and second, the right hand side of any rule may include not only the grammatical terminals and nonterminals but also arbitrary predicates or "tests". The tests must be distinguished from the grammatical vocabulary, and so we mark them by enclosing them in braces, e.g., {test}.

(Pereira and Warren, 1980) define a simple translation which transforms rules like these into Horn clauses in which each n-place nonterminal occurs as a predicate with n+2 arguments. The two added arguments provide a "difference list" representation of the string that is to be parsed under that nonterminal. Given the standard prolog depth-first, backtracking proof technique, these clauses define a standard top-down backtracking parser.

The DCG notation is very powerful. The fact that arbitrary prolog tests are allowed makes the notation as powerful as prolog is: a DCG can effectively parse or recognize exactly the class of effectively parsable or recognizable languages, respectively. Even eliminating the tests would not restrict the power of the system. We get the full power of pure prolog when we are allowed to give our grammatical predicates arbitrary arguments. With just two arguments to grammatical predicates to hold the difference list representation of the string to be parsed, we could recognize only context free languages, but with the extra arguments, it is not hard to define context sensitive languages like $a^nb^nc^n$ which are not context free (cf., Pereira, 1983).

III. EXTRAPOSITION GRAMMARS (XGs)

In spite of the power of DCGs, they are not convenient for the definition of certain constructions in natural languages. Most notable among these are the "movement-trace" or "filler-gap" constructions. These are constructions in which a constituent seems to have been moved from another position in the sentence. This treatment of natural language syntax has been well motivated by recent work in linguistic theory.

For example, there are good reasons to regard the relative pronoun that introduces a relative clause as having been moved from a subject or object position in the clause. In the following sentences, the relative clauses have been enclosed in brackets, and positions from which "who" has moved is indicated by the position of the coindexed "[t]", which is called the "trace":

The woman$_i$ [who [t]$_i$ likes books] reads.
The woman [who$_i$ booksellers like [t]$_i$] reads.
The woman [who$_i$ the bookseller told me about [t]$_i$] reads.

In ATN parsers like LUNAR (Woods, 1970), filler-gap constructions are parsed by what can be regarded as a context free parser augmented with a "HOLD" list: when a prefixed wh-phrase like "in which garage" or "who" is parsed, it is put into the HOLD list from which it can be brought to fill a "gap" in the sentence that follows. Fernando Pereira (Pereira, 1981, 1983) showed how a very similar parsing method could be implemented in logic programming systems. These augmented grammars, which Pereira calls "extraposition grammars" (XGs) allow everything found in DCGs and allow, in addition, rules which put an element into a HOLD list - actually, Pereira calls the data structure which is analogous to the ATN HOLD list an "extraposition list". So, for example, in addition to DCG rules, XGs accept rules like the following:

```
nt ... trace --> RHS
```

where the RHS is any sequence of terminals, nonterminals, and tests, as in DCGs. The left side of an XG rule need not be a single nonterminal, but can be a nonterminal followed by '...' and by any finite sequence of terminals or nonterminals. The last example can be read, roughly, as saying that nt can be expanded to RHS on condition that the category "trace" is given an empty realization later in the parse. We realize nt as RHS and put trace on the extraposition list.

This allows for a very natural treatment of certain filler-gap constructions. For example, Pereira points out that relative clauses can, at first blush, be handled with rules like the following:

```
np --> det , n.
np --> det , n , relative.
np --> trace.

relative --> rel_marker , s.
rel_marker...trace --> rel_pro.
rel_pro --> [who].
```

These rules come close to enforcing the regularity noted earlier: a relative clause has the structure of a relative pronoun followed by a sentence that is missing a noun phrase. What these rules say is that we can expand the relative node to a rel_marker and sentence, and then expand the rel_marker to a relative pronoun on condition that some np that occurs after the relative pronoun be realized as a "trace" that is not realized at all in the terminal string.

It is not hard to see that this set of rules does not quite enforce the noted regularity, though. These rules will allow the relative pronoun to be followed by a sentence that has no gap, so long as a gap can be placed *somewhere* after the relative pronoun. So, for example, these rules would accept a sentence like:

* the woman [who$_i$ the man reads the book] reads [t]$_i$.

In this sentence, a gap cannot be found in the sentence [the man reads the book], but since the second occurrence of "reads" can be followed by an np, we can realize that np as the trace or associated with the moved np "who". But this is clearly a mistake.

To avoid this problem, Pereira suggests treating the extraposition list as a stack, and then "bracketing" relative clauses by putting an element on the stack at the beginning of the relative clause which must be popped off the top before the parsing of the relative can be successfully completed. This prevents filler-gap relationships that would hold between anything outside the relative clause and anything inside.

The rest of this paper does not require a full understanding of Pereira's XGs and their implementation. The important points are the ones we have noted: the extraposition list is used to capture the filler-trace regularities in natural language; and it is used as a stack so that putting dummy elements on top of the stack can prevent access to the list in inappropriate contexts.

IV. RESTRICTED LOGIC GRAMMARS (RLGs)

The XG rules for moved constituents are really very useful. The RLG formalism that will now be presented maintains this feature in a slightly restricted form. RLGs differ from XGs in three respects which can be considered more or less independently. First, RLGs allow a new kind of rules, which we will call "switch rules". Second, we will show how the power of the XG leftward movement rules can be expanded in one respect and restricted in another to accommodate a wider range of linguistic constructions. And finally, we show how a similar treatment allows constrained rightward movement.

A. Switch Rules

In the linguistic literature, the auxiliary verb system in English has been one of the most common examples of the shortcomings of context free grammars. The structure of the auxiliary is roughly described by (Akmajian et al., 1979) in the following way: "The facts to be accounted for can be stated quite simply: an English sentence can contain any combination of modal, perfective have, progressive be, and passive be, but when more than one of these is present, they must appear in the order given, and each of the elements of the sequence can appear at most once." The difficult thing to account for elegantly in a context free definition is that the first in a sequence of verbs can occur before the subject. So for example, we have:

```
I have been successful.
Have I been successful?
```

This is a rather peculiar phenomenon: it is as if the well defined sequences of auxiliaries can "wrap" themselves around the (arbitrarily long) subject np of the sentence.

Most parsers have special rules to try to exploit the regularity between simple declarative sentences and their corresponding question forms. (Marcus, 1980) and (Berwick, 1982), for example, use a "switch" rule which, when an auxiliary followed by a noun phrase is detected at the beginning of a sentence, attaches the noun phrase to the parse tree first, leaving the auxiliary in its "unwrapped", canonical position, so that it can be parsed with the same rules as are used for parsing the declarative forms.

It turns out to be possible to implement a rule very much like Marcus's in logic programming systems. When an auxiliary is found at the beginning of a sentence, its parsing is postponed while an attempt is made to parse an np immediately following it. When that np is parsed it is just removed from the list of words left to parse, leaving the auxiliary verb sequence in its canonical form. We use a notation like the following:

```
s --> switch(aux_verb , np) , vp.
```

The predicate "switch" triggers the special behavior. These switch rules can be implemented very easily and efficiently in prolog (Stabler, 1986ms, 1983). To account properly for the placement of negation, etc. requires some complication in the rules, but this kind of rule with its simple "look ahead" is exactly what is needed.

B. Leftward Movement

When introducing the XG rules above, we considered some rules for relative clauses but not rules for fronted wh-phrases like the one in "In which garage did you put the car?" or the one in "Which car did you put in the garage?". The most natural rules for these constructions would look something like the following:

```
s --> wh_phrase , s.
wh_phrase...pp_trace(wh_feature) -->
    pp(wh_feature).
wh_phrase...np_trace(wh_feature,Case,Agreement)
    --> np(wh_feature,Case,Agreement).

pp --> pp_trace(wh_feature).
np(Case,Agreement) -->
    np_trace(wh_feature,Case,Agreement).
```

If we assume that these rules are included in the grammar along with the XG rules for relative clauses discussed above, then we properly exclude any possibility of finding the gapped wh-phrase inside a relative clause:

```
* What car did the man [who put
    [np_trace] in the garage] go?
* In which garage did the man [who
    put the car [pp_trace]] go?
```

These sentences are properly ruled out by Pereira's "bracketing" constraint.

There are other restrictions on filler-gap relations, though, that are not captured by the bracketing constraint on relative clauses. The following sentence, for example, would be allowed by rules like the ones proposed above:

```
* About what did they burn [the politician's
    book [pp_trace]]?
* Who did I wonder whether she was (np_trace)?
```

These filler-gap relations are unacceptable. How can this filler-gap relation be blocked? We cannot just use another bracketing constraint to disallow filler-gap relations that cross vp boundaries, because that would disallow lots of good sentences like "What did they burn?".

There is a very powerful and elegant set of constraints on filler-gap relations which covers all of these cases and more: they are specified by Chomsky's (Chomsky, 1981) theories of coreference ("binding") and movement ("bounding"). The relevant principles can be formulated in the following way:

(i) A moved constituent must c-command its trace, where a node α c-commands β if and only if α does not dominate β, but the first branching node that dominates α dominates β.

(ii) No rule can relate a constituent X to constituents Y or Z in a structure of the form:

$$\ldots Y \ldots [\alpha \ldots [\beta \ldots X \ldots] \ldots] \ldots Z \ldots,$$

where α and β are "bounding nodes."
(In English, the bounding nodes for leftward movement are s and np.)

The first rule, the *c-command* constraint, by itself rules out sentences like the following:

```
* The computer [which you wrote the program]
    uses np_trace.
* I saw the man who you knew him and I told
    np_trace.
```

since the first branching node that dominates "who" and "which" in these cases is (on any of the prominent approaches to syntax) a node that does not dominate anything after the "him". The second rule, called *subjacency*, rules out sentences like

```
* Who [s did [np the man with np_trace] like]?
* About what [s did they burn [np my book
    [pp_trace]]]?
```

In the first of these sentences, "who" does c-command the trace, but does so across two bounding nodes. In the second of these sentences, notice that the pp_trace is *inside* the np, so that we are not asking about the "burning", but about the content of the book! This is properly ruled out by subjacency.

There is one additional complication that needs to be added to these constraints in order to allow sentences like:

```
Who [s do you think [s I said [s I read
     [np_trace]]]]?
Who [s does Mary think [s you think
     [s I said [s I read [np_trace]]]]]?
```

These "movements" of wh-phrases are allowed in Chomskian syntax by assuming that wh-phrase movements are "successive cyclic": that is, the movement to the front of the sentence is composed of a number of smaller movements across one s-node into its "comp" node.

The implementation of RLG movement rules is quite natural. The trick is just to restrict the access to the extraposition list so that only the gaps allowed by Chomsky's constraints will be allowed by the parser. The c-command restriction can be enforced by indicating the introduction of a gap at the first branching node that dominates the moved constituent, and making sure that the gap is found before the parsing of the dominating node is complete. So, for example, we replace the following three XG rules with two indicated RLG rules:

```
  (XG rules)
relative --> rel_marker , s.
rel_marker...np_trace --> rel_pro.
rel_pro --> [who].

  (RLG rules)
relative <<< np_trace --> rel_pro , s.
rel_pro --> [who].
```

The change from "..." to "<<<" is made to distinguish this approach to constituents which are moved to the left (leaving a trace to the right) from RLG rules for rightward movement. The XG's additional (linguistically unmotivated) category "rel_marker" is not needed in the RLG because the trace is introduced to the extraposition list *after* the first category has been parsed. So the translation of these RLG rules is similar to the translation of XG rules, except that rel_pro's are not treated as gappable nodes, the traces are indexed, and a test is added to make sure that the trace that is introduced to the extraposition list is gone when the last constituent of the relative has been parsed (see Stabler, 1986ms for implementation details).

Subjacency can be enforced by adding an indication of every bounding node that is crossed to the extraposition list, and then changing the access to the extraposition list. Once this is done, it is clear that we cannot just use the extraposition list as a stack: we have introduced the indications of bounding nodes, and we have indexed the traces. The presence of the bounding node markers allows us to implement subjacency with the rule that a trace cannot be removed from a list if it is covered by more than one bounding marker, unless the trace is of a wh-phrase and there is no more than one covering bound that has no available comp argument.

So, to put the matter roughly, access to the RLG extraposition list is less restrictive than access to the XG's in that the c-command and subjacency constraints are enforced. These restrictions allow a considerable simplification in the grammar rules. Note that the XG rules that were shown as examples are comparable in complexity to the RLG rules shown, but the XG rules were incorrect in the crucial respects that were pointed out! The XG rules shown allowed ungrammatical sentences (viz., violations of the subjacency and c-command constraints) that the RLG rules properly rejected. The XG rules that properly rule out these cases would be considerably more complex.

C. Rightward Movement

Although the preceding account does successfully enforce subjacency for leftward movement, no provisions have been made for any special treatment of rightward moved constituents, as in sentences like the following:

```
[The man [t]_i] arrived [who I
    told you about]_i.
*The woman [who likes [the man [t]_i]]
    arrived  [who I told you about]_i.
```

It is worth pointing out just briefly how these can be accommodated with techniques similar to those already introduced.

There are a number of ways to deal with these constructions: (i) The standard top-down left-to-right strategy of "guessing" whether there is a rightward moved constituent would obviously be expensive. Backtracking all the way to wherever the incorrect guess was made is an expensive process, since a whole sentence with arbitrarily many words may intervene between the incorrect guess and the point where the error causes a failure. (ii) One strategy for avoiding unnecessary backtracking is to use lookahead, but obviously, the lookahead cannot be bounded by any particular number of words in this case. More sophisticated lookahead (bounded to a certain number of linguistically motivated consitituents) can be used (cf., Berwick, 1983), but this approach requires a complicated buffering and parse-building strategy. (iii) A third approach would involve special backward modification of the parse tree, but this is inelegant and computationally expensive. (iv) A fourth approach is to leave the parse tree to the left unspecified, passing a variable to the right. This last strategy can be implemented quite elegantly and feasibly, and it allows for easy enforcement of subjacency. This is the approach we have taken. To handle optional rightward movement (extraposition from np), we use rules like the following:

```
s --> np, vp, adjunct.

optional_rel --> rel.
optional_rel >>> ((adjunct-->rel) ; Tree).
```

In these rules, "Tree" is the variable that gets passed to the right. The last rule can be read informally as saying that optional_rel has the structure Tree, where the content of Tree will be empty unless an "adjunct" category is expanded to a rel, in which case Tree can be instantiated to a trace that can be coindexed with rel.

The situation here is more complicated than the situation in leftward movement. In rightward movement, following (Baltin, 1981), we provide a special node for attachment, the "adjunct" node. This violation of the "structure preserving constraint" has been well motivated by linguistic considerations. The adjunct node

is a node that can do nothing but capture rightward moved pp's or relative clauses.*

A second respect in which rightward movement is more complicated to handle than leftward movement is in the enforcement of subjacency. Since in a left-to-right parse, rightward movement proceeds from an embedded gap position to the moved constituent, we must remove boundary indicators across the element in the extraposition list that indicates a possible rightward movement. So to enforce subjacency, we cannot count boundary indicators between the element and the top; rather we must count the boundary indicators that are removed across the element. Subjacency can be enforced only if the element of the extraposition list that carries "Tree" to the right can also mark whether a bounding category has been passed (i.e., when the parse of a bounding category has been completed). Again, the elaboration of the definition of "virtual" required to implement these ideas is fairly easy to supply (see Stabler 1986ms for implementation details).

V. CONCLUSIONS AND FUTURE WORK

Even grammar notations with unlimited expressive power can lack a *graceful* way to define certain linguistic structures, and they can define structures that never occur in human languages. DCGs have universal power, but XGs immediately offer a facility for elegant characterization of the movement constructions common in natural languages. RLGs are one more step in this direction toward a notation for logic grammars that is really appropriate for natural languages. RLGs provide "switch rule" notation to allow for elegant characterization of "inverted" or "wrapped" structures, and a notation for properly constrained movements that defines filler-gap relations for both rightward and leftward movement, even when those relations are not properly nested. Getting these results in an XG would be considerably more awkward, but our approach has shown how a careful handling of the "extraposition list" allows easy enforcement of movement constraints.** A fairly substantial RLG grammar for English has been constructed. It runs efficiently, but the real argument for RLGs is that their rules for movement are much simpler than would be possible if constraints on movement were not automatically enforced.

*These rules for rightward movement are oversimplified. Most linguists follow (Baltin, 1981) and others in assuming that phrases extraposed from inside a VP are attached inside of that VP, whereas phrases extraposed from subject position are attached at the end of the sentence (in the position we have marked "adjunct"). (Baltin, 1981) points out that this special constraint on rightward movement seems to hold in other languages as well, and that we can capture it by counting VP as a bounding category for rightward movement. This approach could easily be managed in the framework we have set up here, though we do not currently have it implemented.

**The MGs of (Colmerauer, 1978), the GGs of (Dahl, 1984) and other systems are very powerful, and they sometimes allow fairly elegant rules for natural language constructions, but they are not designed to automatically enforce constraints: that burden is left to the grammar writer, and it is not a trivial burden.

ACKNOWLEDGMENTS

I am indebted to Janet Dean Fodor, Fernando Pereira and Yuriy Tarnawsky for helpful discussions. (Stabler, 1986ms) provides a more complete discussion of this material, including implementation details as well as more theoretical discussion.

REFERENCES

[1] Akmajian, A., S. Steele, and T. Wasow. "The Category AUX in Universal Grammar." Linguistic Inquiry, 10 (1979) 1-64.

[2] Baltin, M.R. "Strict Bounding." In C.L. Baker and J.J. McCarthy, eds., The Logical Problem of Language Acquisition. MIT Press (1981).

[3] Berwick, R.C. Locality Principles and the Acquisition of Syntactic Knowledge. Ph.D. Dissertation, MIT Department of Computer Science and Electrical Engineering (1982).

[4] Berwick, R.C. "A Deterministic Parser with Broad Coverage." In Proc. 8th IJCAI, 1983.

[5] Berwick, R.C. and Weinberg, A.S. "Deterministic Parsing and Linguistic Explanation." 1985ms, forthcoming.

[6] Chomsky, N. Lectures on Government and Binding. Foris Publications, Dordrecht, Holland, 1981.

[7] Colmerauer, A. "Metamorphosis Grammars." In L. Bolc, ed., Natural Language Communication with Computers. Springer-Verlag (1978).

[8] Dahl, V. "More on Gapping Grammars." In Proc. of the Int. Conf. on Fifth Generation Computer Systems. Tokyo, Japan, 1984.

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

[10] Pereira, F. "Extraposition Grammars." American Journal of Computational Linguistics, 7 (1981) 243-256.

[11] Pereira, F. "Logic for Natural Language Analysis." Technical Note 275, SRI International, Menlo Park, California, 1983.

[12] Pereira, F. and Warren, D.H.D. "Definite Clause Grammars for Natural Language Analysis." Artificial Intelligence 13 (1980) 231-278.

[13] Stabler, E.P., Jr. "Deterministic and bottom-up parsing in prolog." In Proc. of the National Conference on AI, AAAI-83, 1983.

[14] Stabler, E.P., Jr. "Restricting Logic Grammars with Government-Binding Theory." Unpublished manuscript, submitted to Computational Linguistics (1986ms).

[15] Woods, W.A. "Transition Network Grammars for Natural Language Analysis." Communications of the ACM 13 (1970) 591-606.