Restricting Logic Grammars with Government-Binding Theory

Edward P. Stabler, Jr.[1]

Department of Computer Science
Middlesex College
The University of Western Ontario
London, Ontario N6A 5B7 Canada

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.

1 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 that 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 (grammar induction) problem of correctly identifying an arbitrary language in the class. So there are certainly significant practical advantages to formalisms for natural language parsers that 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 ruling out 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 (1981), Berwick (1980), Wehrli (1984) 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 with lookahead, rather than Marcus's deterministic LR(k,t)-like parsing method. This approach to parsing, which has been developed in logic

0362-613X/87/010001-10$03.00

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.

2 Definite Clause Grammars

Definite clause grammars (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 that (in their most straightforward use) define a top-down, backtracking recognizer or parser in prolog.[2]. In the DCG grammar formalism, every rewrite rule must expand exactly one nonterminal.

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 nonterminal, 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.
np  → det , n.
vp  → v.
det → [the].
n   → [man].
n   → [woman].
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 that includes *the woman reads.*

Two additional features provide DCGs with considerably more power; that is, they allow us to define a class of languages that properly includes the class defined by DCGs just described (those with only 0-place grammatical category symbols). 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 that transforms rules like these into Horn clauses in which each 0-place nonterminal occurs as a predicate with two arguments. These two arguments provide a "difference list" representation of the string that is to be parsed under that nonterminal. (DCG 1) is translated into the following Horn clauses, where variable names begin with an uppercase letter:

```
s(L0,L)  :- np(L0,L1) , vp(L1,L).
np(L0,L) :- det(L0,L1) , n(L1,L).
vp(L0,L) :- v(L0,L).
det([the | L],L).
n([man | L],L).
n([woman | L],L).
v([reads | L],L).
```

The first of these clauses can be read declaratively as "what remains when L is taken off the tail of L0 is an s if the list from L0 to L1 is an np, and the list from L1 to L is a vp". The terminals in the rewrite rules are handled differently, since they must actually be present in the string being recognized. So, for example, the clause for *det* says that when the difference between its two list arguments is just the element *the,* we have a *det.* An empty expansion of a category *cat* would be translated into the clause *cat(L,L).*

Given the standard prolog depth-first, backtracking proof technique, these clauses define a standard top-down backtracking parser. To recognize the sentence *the man reads,* for example, we can ask for a proof of the goal *:- s([the,man,reads],[ ]).* The original string gets "consumed" from the front as it is passed to each grammatical predicate that succeeds.

Prolog tests and extra arguments on grammatical predicates are easily accommodated in the translation to Horn clauses: every *n* place DCG nonterminal corresponds to an *n+2* place predicate in the prolog translation, where the last two added arguments hold the difference lists as above; and every test is simply conjoined with the translations of the grammatical predicates without adding any extra arguments.

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 $a^nb^nc^n$ that are not context free (cf., Pereira 1983).

3 Extraposition Grammars

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" constructions. These are constructions in which a constituent seems to have been moved from another position in the sentence. There are well-known traditions in linguistic theory that do not use movement rules at all (cf., e.g., Gazdar et al. 1985), but the

Chomskian tradition ("Government-Binding" theory, and its predecessors) makes crucial use of movement analyses, specifically the rule move-α. It is natural to think of this aspect of Government-Binding theory in terms of the movements of constituents, but as Wasow (1985) and others have noted, the main work done by the movement rules is to relate ("co-index") positions in the structural representations of the sentence. So long as these relations between structural positions are properly constrained, we do not really need to think of the relations as having been established by a movement from an original position (in "d-structure") to a new position (in "s-structure").[3]

This paper will not provide an introduction to Chomskian syntax, but the basic idea behind movement rules is fairly easy to illustrate. There are, for example, 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 co-indexed trace, [t]:

The woman$_1$ [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), constructions produced by movement are parsed by what can be regarded as a context-free parser augmented with a "HOLD" list: when a fronted wh-phrase like *who* is parsed, it can be put into the HOLD list from which it can be brought to fill (or to allow a co-indexed trace to fill) a later position in the sentence. Fernando 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 that put an element into a HOLD list – actually, Pereira calls the data structure 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 movement constructions. For example, Pereira points out that relative clauses can, at first blush, be handled with rules like the following:

```
s → np , vp.
vp → v.
vp → v , np.
np → det , n , optional_relative.
np → trace.
optional_relative → [ ].
optional_relative → relative.
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 a sentence *s*, and then expand the *rel_marker* to a relative pronoun *rel_pro* 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 trace, so long as a trace 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 trace 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 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 that must be popped off the top before the parsing of the relative can be successfully completed. This can be accomplished by changing our rule for relatives to the following:

```
relative → open , rel_marker , s , close.
open...close → [ ].
```

This prevents movements that would relate anything outside the relative clause to anything inside.

3.1 THE IMPLEMENTATION OF XGs

The implementation of XGs is quite straightforward. Pereira translates each nonterminal category symbol with *n* arguments into a predicate with *n+4* arguments: the first two added arguments hold the difference list representation of the string to be parsed (as in DCGs), and the second two additional arguments hold a representation of the extraposition list (one argument has the list "coming in", the other holds the list "going out"). A rule like the last rule defining *open,* translates into two prolog clauses: one that rewrites open as the empty terminal string [ ] and adds *close* to the top of the extraposition list; and

another that says that *close* can be expanded as a "virtual constituent" with no terminal realization at all:

```
open(L,L,X,x(gap,nonterminal,close,X)).
close(L,L,X0,X) :- virtual(close,X0,X).
```

(The functor that connects the elements of the extraposition list is not any standard list constructor, but the functor x.) The predicate virtual used in the rule for close is defined with the single clause:

```
virtual(NT,x(C,nonterminal,NT,X),X).
```

This just says that we can realize a nonterminal NT (in any context C) just by taking it off the top of the extraposition list. Like the lists that represent the string to be parsed, the extraposition lists are passed down from a father to the first sibling, and then from sibling to sibling, and so on to every nonterminal node in the tree. This is more efficient than treating the gaps as features that are passed only to the descendants of a node, because of the fact that a particular moved constituent can (in most cases) correspond to only one trace in a sentence. Thus, if the subject np of a relative clause is trace, the direct object cannot be; if the direct object is trace, the indirect object cannot be; and so on. It makes sense to pass the extraposition list from sibling to sibling rather than just down from the parent, because only one np can fill any particular gap, and often it can be any of a number of categories.[4]

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 movement constructions that can occur in natural language; 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; and the extraposition list is passed to every node of the derivation tree.

4 Restricted Logic Grammars

The XG rules for moved constituents are really very useful. The restricted logic grammar (RLGs) formalism presented now maintains this feature in a slightly restricted form, so the best way to introduce RLGs is to explain how they differ from XGs. They differ 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.

4.1 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 auxiliary shows some striking regularities. The basic idea has been neatly formulated 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." These verbs occur before the main verb of the sentence, of course, but the more 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?
* I been have successful.
* Been I have 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.[5]

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.[6]

Pereira (1983) does not attempt to provide a full grammar for English, but it is interesting that he proposes rules that treat the auxiliary inversion on the model of movement constructions, proposing a rule rather like the following:

```
s → fronted_verb , s.
fronted_verb...aux_verb(Features) →
                                    aux_verb(Features).
```

These rules allow us to find an auxiliary verb followed by a sentence that is missing an auxiliary verb with the same features. This almost captures the regularity we want, but it is too permissive in just the way our first set of XG rules for relative clauses was. These rules would allow us to accept strings like:

Has he [t] been saying that he has been succeeding?
* Has he has been saying that he [t] been succeeding?

The problem is that we want to make sure that the *aux_verb* put on the extraposition list is removed right after the subject is parsed, not later in the sentence. There is no elegant way to use the bracketing technique, because there is no motivated constituent in these sentences that contains the fronted auxiliary, the subject noun phrase, and the rest of the auxiliary verbs. Something rather different is required.

It turns out to be very easy to implement a rule very much like Marcus's inversion rule in logic programming systems. These rules do not put an element in the extra-

position list to be removed sometime before the end of the sentence. Rather, 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 this special behavior: when the first word in the string to be parsed is an *aux_verb*, it is ignored while an attempt is made to parse an np; if the attempt to parse an np is successful, then an attempt is made to parse a vp given the string that has the *aux_verb* as its first element, followed by whatever followed the np. In general, the argument to switch is always a term of the form $test_1$, $test_2$, ..., $test_n$, *cat*, where $test_1$, ..., $test_n$ are tests on features of the first *n* lexical items in the string, and *cat* is a nonterminal to be found in the string that begins with the *n+1st* element of the string to be parsed.

The implementation of switch rules is surprisingly easy.[7] The simple rule given is translated into the following prolog clause:

```
s([First | L0],L,X0,X) :- aux_verb(First) ,
                          np(L0,L1,X0,X1) ,
                          vp([First | L1],L,X1,X).
```

where *aux_verb(First)* just checks the dictionary to see if this first element of the string is an auxiliary verb. A complete treatment of the English auxiliary system (with negation and adverbs, etc.) is more complicated, but this kind of rule with its simple "look ahead" is exactly what is needed. It is even more efficient than the XG approach discussed above.

4.2 LEFTWARD MOVEMENT

When introducing the movement rules of XGs 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:[8]

```
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 trace of a fronted 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 movement, 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]$_i$ did they burn [the politician's book [pp_trace]$_i$]?

* What$_i$ did he ask where I hid [np_trace]$_i$?

* Who$_i$ did I wonder whether she was [np_trace]$_i$?

These movements are unacceptable, but how can they be blocked? We cannot just use another bracketing constraint to disallow movements that cross vp boundaries, because that would disallow good sentences like *What$_i$ did they burn [np_trace]$_i$?*.

There is a very powerful and elegant set of constraints on movement that covers the cases we have considered and subsumes the relative clause island constraint: they are specified by Chomsky's (1981) theories of coreference ("binding") and movement ("bounding"). The "co-indexing" of the moved constituent and its trace marks a "binding" relationship: the trace is coreferential with the moved constituent. The relevant principles can be formulated in the following way:[9]

(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:

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

where α and β are "bounding nodes". (We will assume that the bounding nodes for leftward movement in English are *s* and *np*.)

The first rule, the c-command restriction on binding, suffices by itself to rule out sentences like the following:

* The computer [which$_i$ you wrote the program] uses [np_trace]$_i$.

* I saw the man [who$_i$ you knew him] and I told [np_trace]$_i$.

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**, is a bounding restriction, a restriction on the constituents that can be related by movement. Subjacency rules out sentences like

* Who$_i$ [s did [np the man with [np_trace]$_i$] like]?

* [About what]$_i$ [s did they burn [np my book [pp_trace]$_i$]]?

In the first of these sentences, *who* does c-command the *np_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 also 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$_i$ [s do you think [s I said [s I read [np__trace]$_i$]]]?

Who$_i$ [s does Mary think [s you think [s I said [s I read [np__trace]$_i$]]]]?

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. It is further assumed here that a wh-phrase cannot be moved out of an s that already has a wh-phrase in its *comp*. This allows the last examples, while disallowing cases like the following:

* Who$_i$ did you wonder who$_j$ trace$_i$ saw trace$_j$?
* Who$_i$ did you wonder who$_j$ trace$_j$ saw trace$_i$?

The implementation of RLG movement rules to automatically enforce these constraints is quite natural. In the first place, current syntactic theory does not allow just any constituent to be moved, so we do not need to pass the extraposition list to every node. It only needs to be passed to the nodes that could dominate a trace. For example, the passing of the list to *det*, *n*, and *v* nodes is just wasted effort since none of these categories can be moved or dominate a trace. Allowing only certain categories to carry the extraposition list, where those categories are all nonterminals, simplifies the translation to prolog clauses and makes the resulting parser more efficient. (It also makes it convenient to use standard list notation for the extraposition list.)

The trick then is to restrict the access to the extraposition list so the parser will allow traces only in the positions allowed by Chomsky's constraints. The c-command restriction can be enforced by indicating the introduction of a trace at the first branching node that dominates the moved constituent, and making sure that the trace is found before the parsing of that 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 the XG functor "..." to "<<<" is made to distinguish this approach to parsing constituents that are moved to the left (leaving a trace to the right) from RLG rules for rightward movement. The XGs 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.[10] So the translation of these RLG rules is similar to the XG translation of XG rules, except that *rel__pros* are not passed the extraposition list, 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:

```
relative(L0,L,X0,X) :- rel__pro(L0,L1) ,
                       s(L1,L,trace(Index).X0,X) ,
                       tracegone(trace(Index),X).
```

This enforces the c-command constraint, because everything that is c-commanded by the relative pronoun *rel__pro* is under the relative node.

The RLG grammar compiler can enforce subjacency automatically by giving special treatment to grammar rules that expand bounding categories. All that is required is the addition of an indication of every bounding node that is crossed to the extraposition list, and then changing the XG definition of the predicate *virtual* to allow the appropriate relations across those nodes. The grammar compiler takes care of this by introducing an element *np__bound* into the extraposition list *before* any daughter of an *np* is parsed, and removing it when the *np* is complete, and similarly for *s* nodes. So the following RLG rules would be translated as shown:

```
(RLG rules)
    s → np , vp.
    np → det , n , relative.
(PROLOG translations)
    s(L0,L,X0,X) :-
        np(L0,L1,[s__bound(__) | X0],X1) ,
        vp(L1,L,X1,[s__bound(__) | X]).
    np(L0,L,X0,X) :-
        det(L0,L1) ,
        n(L1,L2) ,
        relative(L2,L,[np__bound | X0],[np__bound | X]).
```

The prolog translation for *s* puts *s__bound(__)* on top of the incoming extraposition list X0, and removes it from the outgoing extraposition list, returning just X. The argument of *s__bound(__)* in this bound indicator is an anonymous variable, __, whose value indicates whether the *comp* node corresponding to the bound has or has had a wh-phrase in it. Since *nps* do not have *comp* positions for elements to move into or through, *np__bound* has no argument.

Now 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 latter point means that the traces will have to be placed not just in any place where a trace is allowed; each trace is uniquely associated with a moved phrase and must be placed in a position where the moved phrase could have come from. For example, it is easy to see that the following co-indexed relationships are not acceptable:

* what$_i$ does the man who$_j$ trace$_i$ reads like trace$_j$?

This sentence with the marked movements is ruled out by subjacency. The same sentence with properly nested co-indexing, on the other hand, is acceptable and is allowed by subjacency.

In any case, it is clear that the extraposition list cannot literally be treated as a stack. 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.

The following rules for virtual are a good first approximation:

```
virtual(NT,[NT | X],X).
virtual(NT,[np__bound,NT | X],[np__bound | X]).
virtual(NT,[s__bound(NT),NT | X],[s__bound(NT) | X]).
virtual(NT,[s__bound(NT) | X],[s__bound(NT) | Y]) :-
                                    wh(NT) , virtual(NT,X,Y).
```

The first of these rules just takes a trace off the top of the list, returning the remainder of the list. The second and third rules allow a trace to be removed from under a single *np__bound* or *s__bound.* The fourth rule allows a trace to be removed from under any number of *s__bounds,* filling the *comp* argument of each with the moved constituent ń to make it unavailable for other wh-phrases.

These rules about access to the extraposition list do not allow the removal of one trace from under another: the traces themselves are available on a strictly last in, first out basis, as if they were in a stack. This has the consequence that moved constituent-trace relations can only be properly nested, as in:

[Which violins]$_i$ are [the sonatas]$_j$ easy to play trace$_j$ on trace$_i$

* [Which violins]$_i$ are [the sonatas]$_j$ easy to play trace$_i$ on trace$_j$

An argument against this restriction on co-indexed relations comes from sentences like the following:[11]

What$_i$ do you know how$_j$ to read trace$_i$ trace$_j$?

Fodor (1983) has argued, though, that this sort of crossing relation can only occur with traces of different categories: in the last example, the crossing relations between an adverb and a noun phrase can occur, but crossing relations between two noun phrases and their traces cannot occur (unless that relation is dictated by subjacency or other constraints). So we must allow one trace to be removed from the list across another *when the traces are of different linguistic categories.* This modification is easily made: the required modification in the definition of virtual is straightforward.

Since the aim of this paper is to turn over to the grammar compiler the enforcement of universal constraints in order to simplify the task of grammar construction, it should be noted that the implementation of subjacency just described, while it may be appropriate for English, is not appropriate for any language in which the bounding nodes are not *s* and *np.* Rizzi (1982) has argued that there is variation among languages in the selection of bounding nodes: in particular, he argues that the bounding nodes in Italian are *s__bar* and *np.* The RLG grammar compiler can easily accommodate this variable parameter: the appropriate bounding nodes just need to be marked so that they can be submitted to the special treatment described here. Similarly, the approach just described requires that the grammar compiler know which categories can dominate a trace. It is easy to accommodate variation here as well. Our current implementation requires that the grammar writer specify what these nodes are, but it would be possible to implement a two-pass grammar compiler that would compute these nodes after its first pass and then do the appropriate compilation in to prolog clauses.[12]

In summary, to put the matter roughly, access to the RLG extraposition list is less restrictive than access to the XGs because the list of traces is not treated as a stack – we allow a trace to be removed from the list across another of a different category; but it is more restrictive in enforcing the c-command and subjacency constraints and because of the restrictions on the nodes at which the extraposition list is available. These restrictions allow a considerable simplification in the grammar rules while preserving enough flexibility to allow for relevant variations among different natural languages.[13]

4.3 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]$_i$.

[What book [t]$_i$] arrived [about the arms race]$_i$?

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

* What woman [who likes [the book [t]$_i$]] arrived [about the arms race]$_i$?

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

It should be noted that some phrase structure approaches do not relate (what we are treating as) rightward moved constituents to any other positions in the sentence structure (leaving that to the semantics), but we will follow the Chomskian tradition in assuming that the syntactic parser should mark this relation. There are a number of ways to do this:

- 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.

- 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 constituents) can be used (cf., Berwick 1983), but this approach requires a complicated buffering and parse-building strategy.
- A third approach would involve special backward modification of the parse tree, but this is inelegant and computationally expensive.
- A fourth approach in left-to-right parsing 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.

To handle optional rightward "extraposition from *np*" using this last strategy, we use rules like the following:

```
s → np, vp, optional__adjunct.
optional__adjunct → [ ].
optional__adjunct → 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 co-indexed 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 can do nothing but capture rightward moved *pp*s or relative clauses.[14]

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 trace 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 have been 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 dominating bounding category has been completed). In optional movement, the crossing of a second bounding category can just instantiate *Tree* to the empty list. We use an element in the extraposition list of the form *right(Rule,Tree,BoundFlag)*. Crossing one bound instantiates *BoundFlag;* crossing a second bound instantiates *Tree* to the empty list and removes the element from the extraposition list. Again, the elaboration of the definition of *virtual* required to implement these ideas is fairly easy to supply.

One approach to implementing a rightward movement rule like the one above is to translate it into two prolog clauses, one to initiate the rightward movement, and one providing the landing site:

```
optional__rel(L,L,X,
   [right((adjunct(S0,S,H0,H) :-
                  rel(S0,S,H0,H)),Tree,Bound) | X]).
adjunct(L0,L,X0,X) :-
              rightward(X0,X1,adjunct(L0,L,X1,X).
```

The head of the *Rule*, which is the first argument of the element *right(Rule,Tree,BoundFlag),* specifies where the moved constituent can "land"; the body of the *Rule* tells us what constituent has been moved. The predicate rightward is defined as part of the grammar interpreter:

```
rightward(X0,X,Cat) :-
          find__constituent(X0,X,(Cat:-RHS)), RHS.
find__constituent(X0,X,(Cat:-RHS)) :-
                X0=[right((Cat:-RHS),trace,__) | X].
find__constituent(X0,[right(R,T,B) | X],(Cat:-RHS)) :-
                X0=[right(R,T,B) | X1],
                find__constituent(X1,X,(Cat:-RHS)).
```

This enforcement of subjacency in rightward movement immediately rules out the two ungrammatical examples shown above.

5 Conclusions and Future Work

Even grammar notations with unlimited expressive power can lack a graceful way to define certain linguistic structures. 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 leftward and rightward movement, even when the resulting bindings are not properly nested. Getting these results in an XG would be considerably more awkward.[15]

A fairly substantial RLG grammar for English has been constructed, and 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. We are exploring the automatic enforcement of more of the principles of government and binding theory. It is unfortunate that efficient implementation of these constraints requires such careful attention to the procedural details of the parsing mechanism. To formalize the problem of implementing these constraints, we are designing a "grammar grammar" that automatically compiles an

elegant, modular, logical statement of grammatical principles into a parser that properly and efficiently enforces them. This work extends the current approach and differs from the approach of Barton (1984), Barton and Berwick (1985), and others in that the constraints are represented explicitly rather than being respected in virtue of the of architecture of the parsing mechanism.

References

Akmajian, A.; Steele, S.; and Wasow, T. 1979 The Category AUX in Universal Grammar. *Linguistic Inquiry* 10: 1-64.

Baltin, M.R. 1981 Strict Bounding. In Baker, C.L. and McCarthy, J.J., Eds., *The Logical Problem of Language Acquisition.* MIT Press, Cambridge, Massachusetts.

Barton, E. 1984 Toward a Principle-Based Parser. A.I. Memo 788, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Barton, E. and Berwick, R.C. 1985 Parsing with Assertion Sets and Information Monotonicity. *Proceedings of the 9th IJCAI.* Los Angeles, California.

Berwick, R.C. 1982 Locality Principles and the Acquisition of Syntactic Knowledge. Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Berwick, R.C. 1983 A Deterministic Parser with Broad Coverage. *Proceedings of the 8th IJCAI.* Karlsruhle, Germany.

Berwick, R.C. and Weinberg, A.S. 1985 Deterministic Parsing and Linguistic Explanation. MS, forthcoming in *Language and Cognitive Processes.*

Chomsky, N. 1981 *Lectures on Government and Binding.* Foris Publications, Dordrecht, Holland.

Colmerauer, A. 1978 Metamorphosis Grammars. In Bolc, L., Ed., *Natural Language Communication with Computers.* Springer-Verlag.

Dahl, V. 1984 More on Gapping Grammars. *Proceedings of the International Conference on Fifth Generation Computer Systems.*

Fodor, J.D. 1983 Phrase Structure Parsing and the Island Constraints. *Linguistics and Philosophy* 6: 163-223.

Gazdar, G.; Klein, E.; Pullum, G.; and Sag, I. 1985 *Generalized Phrase Structure Grammar.* Harvard University Press, Cambridge, Massachusetts.

Manzini, M.R. 1983 On Control and Control Theory. *Linguistic Inquiry* 14(3): 421-446.

Marcus, M. 1980 *A Theory of Syntactic Recognition for Natural Language.* MIT Press, Cambridge, Massachusetts.

Pereira, F. 1981 Extraposition Grammars. *American Journal of Computational Linguistics* 7: 243-256.

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

Pereira, F. 1985 Note on "DCGs and Parsing Strategy". *Prolog Digest* 3(41).

Pereira, F. and Warren, D.H.D. 1980 Definite Clause Grammars for Natural Language Analysis. *Artificial Intelligence* 13: 231-278.

Pollard, C. 1984 Generalized Phrase Structure Grammars, Head Grammars, and Natural Language. Ph.D. dissertation, Stanford University, Palo Alto, California.

Rizzi, L. 1982 *Issues in Italian Syntax.* Foris Publications, Dordrecht, Holland.

Stabler, E.P. Jr. 1983 Deterministic and Bottom-Up Parsing in Prolog. *Proceedings of the National Conference on Artificial Intelligence, AAAI-83.*

Wasow, T. 1985 Postscript. In Sells, P., Ed., *Lectures on Contemporary Syntactic Theories.* Center for the Study of Language and Information, Stanford University.

Wehrli, E. 1984 A Government-Binding Parser for French. Working Paper No. 48, Institut pour les Études Semantiques et Cognitives, Université de Geneve, Geneva, Switzerland.

Woods, W.A. 1970 Transition Network Grammars for Natural Language Analysis. *Communications of the ACM* 13: 591-606.

Notes

1. I am indebted to Janet Dean Fodor, Fernando Pereira, and Yuriy Tarnawsky for helpful discussions of this material. Discussions after a presentation of parts of this material at the University of Toronto in March 1986 also inspired some significant improvements, as did the comments of an anonymous referee. Richard O'Keefe provided valuable advice on aspects of the design of the prolog implementations.
2. The qualification "in their most straightforward use" really is necessary. As noted below, DCGs with tests or with grammatical predicates that have more than two arguments have the full universal power of prolog. It is no surprise, then, that DCGs can be used to define bottom-up parsers, as Pereira (1985) points out. The DCG notation is actually a convenient one for the definition of all sorts of parsers. They provide a convenient representation of the string to be parsed, as we will see.
3. Chomsky (1981, p.33) points this out as well: "It is immaterial ... whether Move-α is regarded as a rule forming s-structure from d-structure, or whether it is regarded as a property of s-structures that are 'base-generated'.... It is in fact far from clear that there is a distinction apart from terminology between these two formulations."
4. Parasitic gaps complicate the story here – hence the parenthetical qualification "in most cases", above. In certain cases a single moved constituent can have two gaps, as in *Which articles did Dana file [t] without reading [t]?.* Gazdar et al. (1985) in fact proposes an analysis according to which any gap can be passed to both the NP and the VP under an S node. Consequently, their grammar accepts some strange things like *Which authors did reviewers of [t] always detest [t]?.* In a practical system, one wants to avoid getting parses that are as unlikely to occur as these. A more restrictive Chomskian analysis of parasitic gaps has been proposed and looks like it may be usable in parsing with methods for leftward movement like those described below. Roughly, when the "real" trace is found, the trick is to put another special operator in the extraposition list that allows a subjacent parasitic gap but doesn't require one (Berwick and Weinberg 1985). (I do not mean to claim that a restrictive GPSG analysis of parasitic gaps could not be formulated – Gazdar et al. (1985) is just an example of a relatively unrestricted analysis.)
5. This sort of rule may be useful for other constructions as well. Pollard (1984) argues that, even just in English, a similar "wrapping" analysis is appropriate for many constructions: the phrase *take to task* seems to wrap around its object in *take Kim to task;* the phrase *much taller than Sandy* seems wrapped into the adjective phrase in *Kim is a much taller person than Sandy;* and a similar wrap analysis is proposed to relate *Kim is very easy to please* and *Kim is a very easy person to please.*
6. Actually, Marcus (1980) used a special subject-auxiliary inversion rule, and Berwick (1982) noted that the effect of Marcus's rule can be achieved with a very simple "switch" mechanism that can be assumed to be one of a small set of primitive parser operations.
7. The "lookahead" technique used here for switch rules is described in a slightly more general form in Stabler (1983).
8. Notice that we need different symbols for traces of different categories, since our trace handling mechanism does not check the identity of the node dominating a trace *[t].*
9. The principles actually proposed in Chomsky (1981) are a little more complex, but the versions formulated here suffice for illustrating the basic approach which can be applied to the more sophisticated formulations. In spite of the simplification, the versions presented here provide the desired simplification of the grammar rules.
10. In any case in which the moved element was not the first sibling under the dominating branching node, a different sort of rule would have to be used. We allow the three place predicate '<<<'(Nt,Trace,N), where Nt can have at most M daughters and N ϵ [1,2,...,M-1]. These rules introduce the Trace into the extraposition list after the Nth daughter of Nt. Since '<<<'(Nt,Trace,1) is the most common case in English, we give it the short form 'Nt<<<Trace'.

11. One other common construction with crossing bindings that Fodor (1983) mentions is illustrated with examples like *Who$_i$ did you$_j$ ask trace$_i$ whether PRO$_j$ to blame yourself$_j$?* As indicated, the subject of the embedded clause in this construction is not a trace produced by a movement but another type of empty category: a base-generated pronominal element which is controlled by the higher subject. A movement analysis would be inappropriate for these constructions (Chomsky 1981; Manzini 1983).
12. A compiler that will do this is under construction, together with a proof of its correctness. A complete account is beyond the scope of this paper, but the idea is easy to see. Basically, on the first pass we construct a CFG corresponding to the RLG being processed, a CFG that would generate the same derivations as the RLG except that it is not restricted by the requirements about the presence of appropriate elements in the extraposition list for the expansion of *np* or of a wh__phrase to a trace. We then compute the nodes that can dominate traces in this CFG, working backwards from the right hand sides of the rules. In specifiable cases (which will typically hold), this set will be exactly the set of nodes that can in fact dominate the traces in RLG derivations; otherwise, it is a superset of the set of nodes that can dominate the traces in RLG derivations (though this superset will typically still be a proper subset of the set of nonterminals, and hence useful).
13. Notice 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.
14. These rules for rightward movement are oversimplified. Most linguists in the Government-Binding tradition 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 roughly 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. Notice that although rightward movement is not structure-preserving on this linguistic approach, the parser rules for this movement are structure-preserving in the trivial sense that they supply the category adjunct just to accommodate these movements.
15. Colmerauer's (1978) MGs, Dahl's (1984) GGs, 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.