Extending Definite Clause Grammar to Handle Flexible Word Order
V. Pericliev and A. Grigorov

MODALYS - A System for the Semantic-Pragmatic Analysis of Modal Verbs
B. Kipper

A System for Text Temporal Information Retrieval
I. P. Rodrigues and J. G. P. Lopes

VI. KNOWLEDGE BASED SYSTEMS - METHODS AND ARCHITECTURES

Meta Theory as a Tool for Integration and Control
H. Vassilev

Systems-Based Knowledge Representation: Relations and Methods
I. Dimitrov

COPE - A Flexible Constraint-Based Programming System for Knowledge Processing
S. Trausan-Matu, M. Barbuceanu and G. Ghiculete

A Two-Headed Architecture for Intelligent Multimedia Man-Machine Interaction
P. Quaresma and J. G. P. Lopes

VII. APPLICATIONS OF KNOWLEDGE BASED SYSTEMS

Discovery Environments for the Domain of Computer Programming: A Methodology
H. Ramadhan
Extending Definite Clause Grammar to handle flexible word order

Vladimir PERICLIEV and Alexander GRIGOROV

Mathematical Linguistics Department, Institute of Mathematics with Computing Centre, bl.8, 1113 Sofia, Bulgaria

Abstract

An efficiently implemented extension to DCG is described, allowing a concise, modular and declarative statement of intricate word order regularities. Besides the linear precedence restriction of GPSG, our formalism defines other restrictions (adjacency and linear position) which may be combined into complex expressions.

1. INTRODUCTION

Word order (WO) variation in natural languages presents significant difficulties to computer systems. To mention but the most obvious, systems using CF-style rules will have to specify an intolerably great number of rules in parsing a free WO language (e.g. for 5 constituents, which may freely permute, the number of rules is 5! = 120); conversely, systems elegantly parsing free WO (e.g. by parsing unordered sets, cf. Johnson 1985), will have to additionally enforce, often with much too powerful mechanisms, restricted permutation. By far the greatest number of world languages, however, stand somewhere in-between (e.g. Steele 1981), and even the proclaimed examples of rigid WO languages (English) exhibit variation, whereas those with proclaimed total scrambling (Warlpiri; cf. Hale 1981) show restrictions (Kashket 1987). Therefore, we need general tools, convenient for processing "flexible" WO, by which we mean complex WO regularities, including both extremes.

In this paper, we present a logic-based formalism (an extension to the Definite Clause Grammars, DCG, (Pereira & Warren 1980), which is capable of accomplishing this task in a concise, modular and declarative way. We build upon the insights of the Immediate Dominance/Linear Precedence (ID/LP) format of Generalized Phrase Structure Grammar (GPSG). Our formalism defines further restrictions on "local trees" (i.e. trees of depth one) like adjacency, linear position, etc. in order to evade some difficulties with the expressive power of ID/LP format. Moreover, the formalism allows these restrictions to be combined into complex logic expressions (with operators like negation, disjunction, etc.) in order to capture the intricate ordering rules of natural languages. The formalism is efficiently implemented, allowing either a compiled or an interpreted mode of operation.

The paper is organized in the following way. In section 2, we briefly look at the WO problem from a logic grammar perspective. Sections 3 and 4 respectively describe the formalism and illustrate its application with a reasonably complex example. Section 5 outlines two efficient implementations, one compiling the rules of the formalism into an object grammar (DCG), similarly to the approach in GPSG, and another one, incrementally interpreting these rules. A comparison of the time-complexity of both approaches is made, showing that, though interpreted, the second approach has efficiency gains over the
compiled one in cases of parsing (almost) absolutely free WO languages. Both of our implementations are much more efficient compared to those, using gaps to handle flexible WO (e.g. Popowich 1986).

2. WORD ORDER AND LOGIC GRAMMARS

There seem to be several requirements that a grammar seriously concerned with WO should (try to) fulfil:

(1) Expressive power. - This is the already mentioned capability of (reasonably) handling complex WO phenomena, or "flexible" WO.

(2) Linguistic felicity. - By this we mean the ability to state concisely and declaratively WO rules, in a way maximally approximating linguistic parlance in such situations.

(3) Modularity. - WO restrictions should be expressed modularly, i.e. linearization and immediate dominance relations should (as far as possible) stay apart in rule statements, for there may be many, and diverse, reasons for wanting them easily modifiable. Thus, paradoxically, our knowledge about WO rules, even in well studied languages, is fuzzy (linguistic sources often only provide directions to the effect that, say, in a free WO language, one can freely invert words/constituents, up to the point of ambiguity); or we may wish to tailor a parser to a specific domain with specific WO, etc. In general, as is well known (e.g. by the insights of the Prague School), WO is dependent on a variety of different factors which makes their statement in a rule once and for all quite a hopeless task.

(4) Generativity. - It is highly recommendable that a system can be used for generation, even if it is originally intended for a parser, since, without this capacity, one feels at a loss about the actual statements one has made, formulating a WO rule of some complexity. This should be known to anyone concerned with such a task (and, perhaps, it is not coincidental that e.g. Kay & Karttunen 1984 have first constructed a generator, and used it as a tool in examining the (WO) rules of their grammar, and only then have converted it into a parser).

(5) Efficiency. - The formalism should be efficiently implementable if practical use is at all envisioned.

Most approaches to WO within the logic grammars paradigm (Dahl & Abramson 1990) center around the notion of a "gap", or "skip", which is an unspecified substring in the sentence. In Gapping Grammars (GGs), for instance (Dahl & Abramson 1984, esp. Dahl 1984), free WO is expressed, roughly, by moving around substrings of the parsed string which are referenced by a gap. It is, however, unnatural to express permutations in this way (though indeed movement phenomena are thus elegantly handled). In effect, GGs generally fail to fulfil in a satisfactory way the above requirements: they are clumsy for expressing flexible WO, WO is not declaratively and modularly expressed, GGs cannot be used for generation and, finally, they are not very efficiently implementable. Another powerful formalism, Contextual Discontinuous Grammar (Saint-Dizier 1988), which overcomes the GGs problems with generative capacity and efficiency is also far from being transparent and declarative in expressing WO (e.g. rules with fixed WO are transformed into free order ones by introducing special rules, containing symbols with no linguistic motivation, etc.). There are also problems with another interesting, and early, attempt, using "floating" terminals (Bien et al. 1980).

3. FLEXIBLE WORD ORDER GRAMMAR

Flexible word Order Grammar (FOG) builds upon the ID/LP format of GPSG (Gazdar & Pullum 1981, Gazdar et al. 1985). In GPSG, the two types of information, constituency (=immediate dominance) and linear order, are separated, and local-tree WO rules are concisely, declaratively and modularly expressed. E.g. an immediate dominance rule of the
type \texttt{A --> B D C}, if no linearization restrictions are declared, stands for the mother node expanded into its siblings appearing in any order; declaring the restriction \{ \texttt{D < C} \} e.g., it stands for the rules \{ \texttt{A --> B D C}, \texttt{A --> D B C} \texttt{and A --> D C B} \}.

It is important to note that in GPSG the linear precedence rules stated for a pair of constituents should be valid for the whole set of grammar rules in which these constituents occur, and not just for some specific rule (this "global" empirical constraint on WO is called the Exhaustive Constant Partial Ordering (ECPO) property). ECPO, however, is problematic for describing complex WO variation in natural languages (cf. next section).

FOG therefore extends the expressive power of the LD/LP format, and introduces further WO restrictions which are moreover valid not "globally", i.e. for the whole grammar, but are attached to each specific immediate dominance rule.

FOG allows two types of rules. The first type are the usual DCG rules, used to express fixed WO. On the other hand, FOG employs rules with a double arrow (\( \Rightarrow \)) for non-fixed WO, having the format:

\[
A \Rightarrow \Rightarrow \text{ Constituents} /\text{WO Constraints} \quad \text{or} \quad A \Rightarrow \Rightarrow \text{ Constituents}
\]

where \( A \) is a single non-terminal, \textit{Constituents} is a meta-variable, standing for a set of constituents (set elements being separated by commas), \textit{WO Constraints} is a meta-variable for a logical expression of WO constraints, and \( //\) is a divisor symbol. Rules with WO constraints omitted encode absolutely free WO.

In the immediate dominance part of rules, as in DCG, non-terminals may have arguments, and there are allowed Prolog calls in curly brackets \{\}, interspersed between constituents. In the WO constraint expressions, constituents are referenced only by their functors.

E.g. the following FOG rule will be equivalent to the already mentioned one in GPSG (except that, unlike in GPSG, no claim is made about the order of \( d \) and \( c \) in other grammar rules):

\[
a \Rightarrow \Rightarrow b, c, d // d < c.
\]

(Furtheron, small letters denote Prolog constants, and capital letters Prolog variables.)

In the current implementation(s) we have defined the following atomic WO constraints: 
\begin{itemize}
  \item \textit{Precedence constraints:}
  \begin{itemize}
    \item precedes (e.g. \( a < b \))
    \item immediately precedes (\( a \ll b \)) (we also maintain the notation, \( > \) and \( >> \), for (immediately) follows; see commentary below)
  \end{itemize}
  \item \textit{Adjacency constraints:}
  \begin{itemize}
    \item is adjacent (\( a \ll b \))
  \end{itemize}
  \item \textit{Position constraints:}
  \begin{itemize}
    \item is positioned (\( a \text{ at } N \)), where \( N \) is an integer; e.g. \( a \text{ at } 1 \) designates that \( a \) is constituent-initial.
  \end{itemize}
\end{itemize}

We also allow atomic WO constraints to combine into complex logical expressions, using the following operators with obvious semantics:
\begin{itemize}
  \item Conjunction (notated: \textit{and})
  \item Disjunction (\textit{or})
  \item Negation (\textit{not})
  \item Implication (iff, e.g. \( (b \Rightarrow a) \iff (a \text{ at } 1) \))
  \item Equivalence (iff, e.g. \( (b \Rightarrow a) \iff (a \text{ at } 1) \))
\end{itemize}

Our WO restriction language is, of course, partly logically redundant (e.g. immediately precedence may be expressed through precedence and adjacency, and so is the case with the last two of the operators). However, what is logically is not necessarily psychologically equivalent, and our goal has been to maintain a linguist-friendly notation (cf. requirement 2
of Section 2). To take just one example, we have 'after' in addition to 'before', since linguists normally speak of precedence of dependent with respect to head word, not vice versa, and hence will use both expressions in respective situations (surely it is not by chance that NLSs also have both words).

As a very simple example of how FOG may be used we may consider the grammar of the Latin sentence:

\textit{Puella bona puerum parvum amat}

(good girl loves small boy)

This sentence, which is grammatical in all its permutations, viz. 51, and, therefore, may have discontinuity in the noun phrases, we capture in a single FOG rule with no WO restrictions:

\textit{sentence} \rightarrow \textit{adj nom, noun nom, verb, adj acc, noun acc}.

accompanied by the dictionary rules:

\begin{align*}
\textit{verb} & \rightarrow [amat]. \\
\textit{adj nom} & \rightarrow [bona]. \\
\textit{adj acc} & \rightarrow [parvum]. \\
\textit{noun nom} & \rightarrow [puella]. \\
\textit{noun acc} & \rightarrow [puerum].
\end{align*}

By way of comparison, the GG treatment of this sentence (Dahl 1984), which is also very concise, uses 4 (non-dictionary) rules, with a gapping rule, imitating a permutation, used for each preterminal, adj, noun and verb. What is more important however is that, unlike in GGs, where only absolutely free or rigid WO is neatly expressible, we can unproblematically impose finer-grained restrictions.

As another toy example, if we want to express the WO Universal 20 (of Greenberg and Hawkins) to the effect that NPs comprising dem(onstrative), num(eral), adj(ective) and noun can appear in that order, or in its mirror-image, we can write a "universal" rule enforcing adjacent permutations of all constituents:

\textit{np} \rightarrow \textit{dem, num, adj, noun} // \textit{dem} < > \textit{num and num} < > \textit{adj and adj} < > \textit{noun}.

4. AN EXAMPLE FROM BULGARIAN

Below we show the expressive power of FOG by considering an example from Bulgarian, a Slavic language with quite complex WO regularities.

Bulgarian normally allows quite free ordering of the major constituents, and in our description of Bulgarian we do not recognize a VP node, thus "flattening" the sentence structure, with all types of nominals becoming siblings of the verb. To simplify the discussion, however, below we confine ourselves to just sentences with three siblings, a nominal subject), a verb and an adv(erbial).

Our problem will be to describe three-word declarative sentences, containing a reflexive or non-reflexive verb, as well as their interrogative (yes-no) fellows. It should be clear that the posted orderings, being attached to specific rules, are valid for just these rules (and not necessarily for the whole grammar of Bulgarian).

Declarative sentences with non-reflexive verbs (e.g. Ivan dojde vcerya 'John came yesterday') occur in all six orders, so we have a rule with no restrictions:

\textit{sentence} \rightarrow \textit{nom, verb(V), adv}. \hspace{1cm} (1)

A reflexive verb comprises a verb base plus the reflexive particle "se". The WO restrictions are: (i) the particle "se" (as some other Bulgarian clitic particles) cannot occur
sentence-initially; (ii) the particle immediately precedes the verb, unless the verb is positioned sentence-initially; and (iii) if the verb occurs first, the particle immediately follows it. The other words permute freely as in (1). The FOG translation is:

\[
\text{sentence} \rightarrow \text{nom, refl\_verb(V), adv, refl\_part(se)} \\
\quad \text{// not(refl\_part at 1) and} \\
\quad \text{(refl\_part < < refl\_verb if)} \\
\quad \text{(not(refl\_verb at 1) and)} \\
\quad \text{(refl\_part > > refl\_verb if)} \\
\quad \text{refl\_verb at 1).} \\
\]

(2)

Bulgarian yes-no questions are formed by introducing the interrogative particle "li". The particle is enclitic and occurs immediately after each questioned constituent.

In sentences with non-reflexive verbs the WO constraints are: (i) the question particle, being an enclitic, cannot occur initially; and (ii) if the particle is final, the verb is immediately before it. The FOG translation is again straightforward:

\[
\text{q\_sentence} \rightarrow \text{nom, verb(V), adv, q\_part(li)} \\
\quad \text{// not(q\_part at 1) and} \\
\quad \text{(verb < < q\_part if q\_part at 4).} \\
\]

(3)

In sentences with reflexive verbs the WO constraints are more complicated: (i) neither the reflexive nor the interrogative particle can be initial; (ii) the interrogative and the reflexive particle are contiguous and occur only in this order; and (iii) this particle sequence, as a group, is positioned adjacent to the reflexive verb. In FOG this can be expressed e.g. as:

\[
\text{q\_sentence} \rightarrow \text{nom, refl\_verb(V),} \\
\quad \text{adv, refl\_part(se), q\_part(li)} \\
\quad \text{// not(q\_part at 1) and} \\
\quad \text{(refl\_verb < < refl\_part and)} \\
\quad \text{or (refl\_verb > > refl\_part or)} \\
\quad \text{refl\_verb < < q\_part).} \\
\]

(4)

This example should serve to illustrate that in FOG fairly complex WO regularities can be elegantly expressed. Ordering rules of similar complexity, of course, occur in many other languages with flexible WO besides Bulgarian. Note that FOG can be profitably used even in a rigid WO language like English. E.g. in sentences with "floating" adverbials like "only", "then", etc. (cf. Only he went there, He only went there, He went only there, He went there only), we should only fix the order of major constituents and allow the adverbial to freely permute between them (of course, here we leave questions of scope aside).

We cannot enter into details here, but it would be clear that in the standard ID/LP format some of the WO regularities above will be hard to state, or even totally inexpressible. For one thing, it is seen that constituent order in languages is not necessarily global: a pair of constituents may have one order in one rule, but a different one in another rule, which contradicts ECPQ (for problems with GPSG treatment of so called "complex fronting" in German, cf. Uszkoreit 1985).

5. TWO IMPLEMENTATIONS

The formalism proposed has been implemented in two ways, using a compiled and an interpreted approach. In the compiled implementation, all FOG rules are translated into
DCG rules, spelling out all admissible permutations; e.g. the FOG rule (3) above will result in 14 DCG rules, some of which are given below:

\[
\begin{align*}
q\_sentence & \rightarrow nom, verb(V), q\_part(ii), adv. \\
q\_sentence & \rightarrow nom, adv, verb(V), q\_part(ii). \\
q\_sentence & \rightarrow nom, adv, q\_part(ii), verb(V). \\
q\_sentence & \rightarrow nom, q\_part(ii), verb(V), adv. \\
q\_sentence & \rightarrow nom, q\_part(ii), adv, verb(V). \\
q\_sentence & \rightarrow verb(V), q\_part(ii), nom, adv. \\
q\_sentence & \rightarrow verb(V), q\_part(ii), adv, nom. \\
\end{align*}
\]

The expanded definite clause "object grammar" is then parsed by Prolog's proof mechanism. This results in a relatively efficient parser. However, in "very" free WO languages, this approach may lead to a much too great number of compiled rules. Also, the efficiency is further decreased due to the Prolog's blind backtracking; thus e.g. parsing an input which does not begin with nom according to the above grammar, and using the standard Prolog proof procedure, will result in five times repeating an unsuccessful attempt to parse nom sentence Initially.

To at least partly handle these problems, we have developed an interpreted implementation, using an algorithm in some steps reminiscent of that of Shieber (1984) (A detailed comparison with this, and other known related algorithms, e.g. Evans 1987 et. al., is beyond the scope of this contribution; suffice it to say that we do not use a chart, as in all those approaches, to book-keep partial analyses.)

The algorithm in question does not compile FOG rules into an object DCG to be later parsed, but rather each FOG rule is translated into a Prolog clause, incrementally constructing and executing this rule.

Let \( C_1, C_2, \ldots, C_n \) be the constituents of one FOG rule. Then our parsing algorithm will execute this rule \( n \) steps. Informally, at each step \( k \), where \( k = 1, 2, \ldots, n \), if the already constructed and parsed partial constituent ordering of this FOG rule is \( C_{1k}, C_{2k}, \ldots, C_{nk} \), it is proceeded with the following sub-steps:

\(<1>\) an as yet unparsed constituent, \( C_{ik} \), is selected from the ID part of the FOG rule;

\(<2>\) a check is made whether the newly constructed partial ordering, with \( C_{ik} \) added, viz. \( C_{1k}, C_{2k}, \ldots, C_{ik}, \ldots, C_{nk} \), does not violate the WO constraints stated in this rule;

\(<3>\) an attempt is made to parse the selected constituent \( C_{ik} \) at this position \( k \).

Upon successful parsing of \( C_{ik} \), it is proceeded with the next step \( k+1 \). Upon failure of either sub-step \(<2>\) or \(<3>\) at step \( k \), the algorithm backtracks to sub-step \(<1>\) to select a new constituent \( C_{ik} \), if possible. If this is impossible, the algorithm backtracks to step \( k-1 \).

E.g. the same FOG rule (3) results in the following Prolog clause:

\[
q\_sentence(50, S) :- \\
predicate(\{nom, verb(V), adv, q\_part(ii)\},[]), \\
\text{not(q\_part at 1)}, verb < < q\_part or not(q\_part at 4), \\
S_0, S).
\]

where the definition of \text{parse} (simplified for expository reasons) is as follows:

\[
\begin{align*}
\text{parse}([], \text{OrderConstraints}, S, S) & : = !, \text{OrderConstraints} = [], \\
\text{parse}(\text{Constituents}, \text{ParsedConstituents}, \text{OrderConstraints}, S_0, S) & : = !, \\
\text{select}(\text{Constituent}, \text{Constituents}, \text{RestConstituents}), \\
\text{consistent}(\text{Constituent} | \text{ParsedConstituents}), \\
\text{OrderConstraints}, \text{RestConstituents}), \\
\text{Constituent} = \ldots \{\text{Functor} | \text{Args}\},
\end{align*}
\]
append(Args, [S0, S1], NewArgs),
Goal =.. [Functor | NewArgs],
call(Goal),
parse(RestConstituents, [Functor | ParsedConstituents],
RestConstraints, S1, S).

The predicate parse has 5 arguments. The first one, Constituents, stands for the list of constituents to be parsed. ParsedConstituents is the list of already parsed ones. The third argument, OrderConstraints, is the list of WO constraints (in clausal form), whereas the last two arguments, S0, S, stand for the difference lists.

The predicates select and consistent recursively select a constituent and check WO consistency so far, and after appending the corresponding difference list arguments to this constituent, which results in the goal Goal, in this Goal is executed. The whole process is recursively applied. The complete listing of the parsing program is given in the Appendix.

As should be clear, a basic advantage of this algorithm over the compiled approach (evading the compilation of a possibly huge object grammar aside) is that, unlike Prolog executing a compiled grammar, each constituent here is tried to be parsed only once at an admissible position. Returning to the situation described above with Prolog making 5 unsuccessful attempts to parse nom sentence-initially, the proposed algorithm will, in effect, try to do this superfluous task only once.

For both approaches, time-complexity estimations have been made, relatively to the number of activated Prolog goals, in the case of absolutely free WO (i.e. where there are no WO constraints). The results are given in the table below (N designates the number of constituents per rule):

<table>
<thead>
<tr>
<th></th>
<th>Worst-case</th>
<th>Best-case</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiled implementation</td>
<td>1!+2!+...+N!</td>
<td>N</td>
<td>(N+1)!+2!+...N! )/2</td>
</tr>
<tr>
<td>Interpreted implementation</td>
<td>N*(N+1)/2</td>
<td>N</td>
<td>N*(N+3)/4</td>
</tr>
</tbody>
</table>

The above estimations show that the time-complexity of the compiled approach tends to be exponential, whereas that of the interpreted one to be polynomial in the number of activated goals per rule. Hence, though interpreted, the second approach may enjoy a considerable advantage in efficiency, especially in parsing (almost) absolutely free WO languages.

A further comparison, now with data from Popovich (1986), concerning the performance of his version of GG, FIGG, in cases of flexible WO grammars, shows that both our approaches result into programs which are of more than one order faster, though run on smaller computers. This could be (partly) attributed to the treatment of permutations with the much too powerful notion of gaps.

We also note that our implementations allow FOG grammars to be usable for both analysis and generation (cf. point (4) of Section 2).
6. CONCLUSION

Logic grammars have generally failed to handle flexible WO in a satisfactory way. We have described a formalism which allows the grammar-writer to express complex WO rules in a language in a concise, modular and natural way. FOG extends the expressive power of the ID/LP format of GPSG, and is efficiently implemented to allow its use as a practical tool.

7. REFERENCES


APPENDIX

Listing of the predicate "parse"

parse([], _, OrderConstraints, S, S) :- !, OrderConstraints = [].
parse([Constituent], ParsedConstituents, OrderConstraints, S0, S) :-
select(Constituent, Constituents, RestConstituents),
\[ islist(Constituent). \]
% Constituent is terminal
Functor = Constituent,
consistent(OrderConstraints, [Constituent | ParsedConstituents],
RestOrderConstraints),
append(Constituent, S1, S0)
\;
not(islist(N)),
Constituent =.. [Functor | Args],
consistent(OrderConstraints, [Functor | ParsedConstituents],
RestOrderConstraints),
append(Args, [S0, S1], NewArgs),
Goal =.. [Functor | NewArgs],
call(Goal)
\),
parse(RestConstituents, [Functor | ParsedConstituents],
RestOrderConstraints, S1, S).

consistent([], []).
consistent([Constraint | Constraints], Constituents, RestConstituents) :-
satisfied(Constraint, Constituents),
\!,
consistent(Constraints, Constituents, RestConstituents).
consistent([], _), :-
satisfied(Constraint, Constituents),
\!, fail.
consistent([Constraint | Constraints], Constituents, [Constraint | RestConstituents]) :-
satisfied(Constraint, Constituents, RestConstituents).

satisfied(not(A), Constituents) :- !, unsatisfied(R, Constituents).
satisfied(A1 or A2, Constituents) :- !, unsatisfied(A1, Constituents),
satisfied(A2, Constituents).
satisfied(G > A, [G | Constituents]) :- !, member(A, Constituents).
satisfied(A < < G, [G, A | J]) :- !.
satisfied(G >> > A, [A, G, A | J]) :- !.
satisfied(G << < A, [G, A | J]) :- !.
satisfied(G >> > A, [A, G, A | J]) :- !.
satisfied(G at N, [G | Constituents]) :- !, length([G | Constituents], N).
unsatisfied(not(A), Constituents) :- !,
satisfied(A, Constituents).
unsatisfied(A1 or A2, Constituents) :- !,
satisfied(A1, Constituents),
unsatisfied(A2, Constituents).
unsatisfied(A < G, [G|Constituents]) :- !,
not(member(A, Constituents)).
unsatisfied(G > A, [G|Constituents]) :- !,
not(member(A, Constituents)).
unsatisfied(A < < G, [G|Constituents]) :- !,
Constituents = [A|__].
unsatisfied(G >> A, [G|Constituents]) :- !,
Constituents \= [A|__].
unsatisfied(A < > G, [G,B|Constituents]) :- !,
A \= B,
members(A, Constituents).
unsatisfied(G < > A, [G,B|Constituents]) :- !,
A \= B,
members(A, Constituents).
unsatisfied(G at N, [G|Constituents]) :- !,
length([G|Constituents], N),
N \= N1.
unsatisfied(G at N, [A|Constituents]) :- !,
length([A|Constituents], N).

islist([]).
islist([_|__]).
select(H,[H|T],T).
select(X,[H|T],([H|T]|T1)) :-
select(X,T,T1).
append([],L,L).
append([H|T],L,([H|T]|T1)) :-
append(T,L,T1).
member(H,[H|__]).
member(X,[|__]) :- member(X,T).
length([],0) :- !.
length([_|L,N] :-
length(L,N1),
N = N1 + 1.