Programming in Logic with Constraints for Natural Language Processing

Patrick Saint-Dizier
LSI Université Paul Sabatier
118 route de Narbonne
31062 TOULOUSE Cedex France

Abstract

In this paper, we present a logic-based computational model for movement theory in Government and Binding Theory. For that purpose, we have designed a language called DISLOG. DISLOG stands for programming in logic with discontinuities and permits to express in a simple, concise and declarative way relations or constraints between non-contiguous elements in a structure. DISLOG is also weel adapted to model other types of linguistic phenomena like Quantifier Raising involving long-distance relations or constraints.

1. Introduction

Many areas of natural language syntax and semantics are a fruitful source of inspiration for computer languages and systems designers. The complexity of natural language and the high level of abstraction of most linguistic and semantic theories have motivated the emergence of highly abstract and transparent programming languages. One of the most striking examples is undoubtedly Prolog, initially designed for natural language parsing, via Metamorphosis Grammars (Colmerauer 1978).

For a few years, the Logic Programming paradigm has been augmented with a number of technical and formal devices designed to extend its expressive power. New logic programming languages have emerged, several of them motivated by natural language processing problems. Among them let us mention: CIL (Mukai 1985), designed to express in a direct way concepts of Situation Semantics, MOLOG (Farinas et al. 1985), an extension to Prolog designed to specify in a very simple and declarative way the semantics of modal operators and λ-Prolog (Nadathur and Miller 1988), designed to deal with λ–expressions and λ-reduction.

Recently, the Logic Programming paradigm has been augmented with the concept of *constrained logic programming* (CLP). The basic research done within this area amounts to specifying tools for a more refined control on the type of values or terms a variable in a program can stand for. Answers to goals can be intensional: they are sets of equations (constraints) rather than mere values. Furthermore, the idea at the operational level, incorrect assignments are filtered out as soon as they are encountered when building a proof, making thus proof procedures more efficient.

In this document, we deal with a new, original, type of CLP mechanism: constraints on proof trees. This type of constraint has emerged from, in particular, the definition of a computational model for the quantifier raising operation and for movement theory in Government and Binding theory (noted hereafter as GB). We model those phenomena in terms of constraints between non-contiguous elements in a structure. For example, we want to express constraints between a moved constituent and its co-indexed trace. Constraints are expressed in terms of relations between subtrees in a syntactic tree or in terms of relations between parenthetized constituents in the now more commonly adopted annotated surface forms of sentences.

We have designed Dislog, programming in logic with discontinuities, which permits to express relations between non-contiguous elements in a structure in a simple, declarative and concise way. Dislog is an extension to Prolog; its procedural and declarative semantics are given in (Saint-Dizier 1988b), computer applications like compiler writing and planning are given in (Saint-Dizier 1988a), its use in natural language parsing for free-phrase order languages is given in (Saint-Dizier 1987). In the present document we will focus on modelling movement theory in GB (Chomsky 1982, 1986) and Quantifier Raising (May 1986), which have been in the past two years our main guidelines to specify Dislog. We do not have in mind to build a complete model of GB theory, but we feel that the transfer of some of its main principles and results to the field of natural language processing is worth investigating and is very promising for reasons we will develop hereafter. We are also convinced that GB principles should be paired with other approaches of AI to deal, for example, with the lexicon, lexical semantics, feature representation and control systems and, finally, logical form construction.

2. Movement Theory in GB

In this section, we briefly summarize the main aspects of movement theory (Chomsky 1982, 1986) and give several examples. GB theory is a complete revision of the baroque set of rules and transformations of the standard theory, achieving a much greater expressive power and explanatory adequacy. GB theory is composed of a very small base component (which follows X-bar syntax), a single movement rule and a small set of principles whose role is to control the power of the movement rule. GB exhibits a greater clarity, ease of understanding and linguistic coverage (in spite of some points which remain obscure). The recent formalization of GB theory has several attractive properties for the design of a computational model of natural language processing, among which:

- **concision and economy of means,**
- **high degree of parametrization,**
- **modularity** (e.g. independence of filtering principles),
- **declarativity** (e.g. no order in the application of rules),
- **absence of intermediate structures** (e.g. no deep structure).

GB theory postulates four levels: **d-structure** (sometimes not taken into account, like in our approach), **s-structure** (surface form of structural description), **phonetic form** (PF) and **logical form** (LF). The latter two levels are derived independently from s-structure. We will be mainly interested here in the s-structure level. Movement theory being also applicable, with different parameter values, to LF, we will also show how our approach is well-adapted to characterize LF level from s-structure level.

2.1 Move-α and constraints

In GB, grammaticality of a sentence is based on the existence of a well-formed annotated surface form of that sentence. Thus, no real movements of constituents occur and additional computational and representational problems are avoided. Up to now very few and only partial works have been undertaken to model principles of GB theory. Among them, let us mention (Berwick and Weinberg 1986), (Stabler 1987) and (Brown et al. 1987). There is however an increasing interest for this approach.

GB theory postulates **a single movement rule, more-α,** controlled by principles and filters. This very general rule states:

Move any constituent a to any position.

The most immediate constraints are that α is moved to the left to an empty position (a subject position which is not θ-marked) or is adjoined to a COMP or INFL node (new positions are created from nothing, but this not in contradiction with the projection principle). Constraints and filters control movement but they also force movement. For example, when a verb is used in the passive voice, it can no longer assign case to its object. The object NP must thus move to a place where it is assigned case. The (external) subject θ-role being also suppressed, the object NP naturally moves to the subject position, where it is assigned case, while keeping its previous θ-role.

Another immediate constraint is the **θ-criterion:** each argument has one and only one θ-role and each θ-role is assigned to one and only one argument. Such roles are lexically induced by means of the projection principle (and by lexical insertion), confering thus an increasing role to lexical subcategorization. Finally, **government** gives a precise definition of what a constituent can govern and thus how the projection principled is handled.

Move-α is too abstract to be directly implementable. It needs to be at least partially instantiated, in a way which preserves its generality and its explanatory power. In addition, while the theory is attaining higher and higher levels of adequacy, the interest for analysing the specifics of particular constructions is decreasing. As a consequence, we have to make explicit elements left in the shade or just neglected. Finally, the feature system implicit in GB theory has also to be integrated.

2.2 Examples of movements

All the examples given below are expressed within the framework of head-initial languages like French and English. Let us first consider the **relative clause construction.** In a relative clause, an N is pronominalized and moved to the left and adjoined to a COMP node. A trace of N is left behind and co-indexed with the pronominalized N:

(1) $[_{COMP}$ N(+Pro)$_i$ .... $[_{N2}$ *trace* $_i$] ......]
as in:
$[_{COMP}$ that$_i$ John met $[_{N2}$ *trace* $_i$] yesterday]
Where i is the co-indexation link.

The case of the **passive construction** is a little more complicated and needs to be explained. An object NP is moved to a subject position because the passivisation of the verb no longer allows the verb to assign case to its object NP and a θ-role to its subject NP (in an indirect manner):

at d-structure we have, for example:

[[$_{NP}$] [$_{INFL}$ gives [$_{N2}$ a book]]]

and at s-structure we have:

[[$_{NP}$ a book$_i$] [$_{INFL}$ is given [$_{N2}$ trace$_i$]].

At d-structure, the subject NP is here not mentioned. In a passive construction, the subject is not moved to a PP position (by N2). θ-roles are redistributed when the verb is passivized (this illustrates once again the prominent role played by the lexical description and the projection principle) and a by-complement with the previous θ-role of the subject NP is created.

Another example is the subject-to-subject raising operation, where:

It seems that Jane is on time
becomes:
Jane seems to be on time.

Jane moves to a position without θ-role (it is not θ-marked by *seem*). When the clause *is on time* is in the infinitive form then the subject NP position is no longer case-marked, forcing *Jane* to move:

[$_{INFL}$ Jane$_i$ seem [$_{COMP}$ trace$_i$ [$_{VP}$ to be on time]]]

Finally, let us consider the wh-construal construction occuring at logical form (LF) (May 86). The representation of:

Who saw what ?

is at s-structure:

[$_{COMP2}$ [$_{COMP}$ who$_i$] [$_{INFL}$ trace$_i$ saw [$_N$ what]]]

and becomes at LF:

[$_{COMP2}$ [$_{COMP}$ what$_j$] [$_{COMP}$ who$_i$]] [$_{INFL}$ trace$_i$ saw trace$_j$]].

Both *what* and *who* are adjoined to a COMP node.

This latter type of movement is also restricted by a small number of general principles based on the type of landing site a raised quantifier may occupy and on the nature of the nodes a quantifier can cross over when raised. The first type of constraint will be directly expressed in rules by means of features; the latter will be dealt with in section 5 devoted to Bounding theory, where a model of the subjacency constraint is presented.

2.3 Towards a computational expression of movements

Movements have to be expressed in a simple computational way. Let us consider the relative clause construction (wh-movement in general), all the other examples can be expressed in the same way.

Relative clause construction can be expressed in a declarative way by stating, very informally, that: *within the domain of a COMP, an N(+Pro) is adjoined to that COMP and somewhere else in that domain an N2 is derived into a trace co-indexed with that N(+Pro).* The notion of domain associated to a node like COMP refers to Bounding theory and will be detailed in section 5, the constraint on the co-existence in that domain of an N(+Pro) adjoined to a COMP and, somewhere else, of an N2 derived into a trace can directly be expressed by constraints on syntactic trees, and, thus, by constraints on proof trees in an operational framework. This is precisely the main motivation of DISLOG that we now briefly introduce.

3. An Introduction to DISLOG, Programming in Logic with Discontinuities.

Dislog is an extension to Prolog. It is a language composed of Prolog standard clauses and of Dislog clauses. The computational aspects are similar to that of Prolog. Foundations of DISLOG are given in (Saint-Dizier 1988b). We now introduce and briefly illustrate the main concepts of Dislog.

3.1. Dislog clauses

A Dislog clause is a finite, unordered set of Prolog clauses f_i of the form:

{ f_1 , f_2 , ... , f_n }.

The informal meaning of a Dislog clause is: *if a clause f_i in a Dislog clause is used in a given proof tree, then all the other f_j of that Dislog clause must be used to build that proof tree, with the same substitutions applied to identical variables.* For example, the Dislog clause (with empty bodies here, for the sake of clarity):

{ *arc(a,b)* , *arc(e,f)* }.

means that, in a graph, the use of *arc(a,b)* to construct a proof is conditional to the use of *arc(e,f)*. If one is looking for paths in a graph, this means that all path going through *arc(a,b)* will also have to go through *arc(e,f)*.

A Dislog clause with a single element is equivalent to a Prolog clause (also called definite program clause).

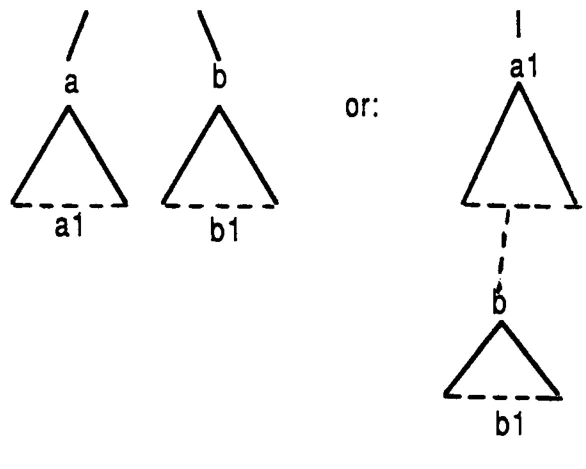
A Dislog program is composed of a set of Dislog clauses. The definition of a predicate p in a Dislog program is the set of all Dislog clauses which contain at least one definite clause with head predicate symbol p. Here is an example of a possible definition for p:

{ p(1) , h :- t(X) }.
{ (p(X) :- t(X), p(X-1)) , d(3) }.
{ p(5) }.

A full example is given in section 3.3.

3.2 Constraining Dislog clauses

We now propose some simple restrictions of the above general form for Dislog clauses. A first type of restriction is to impose restrictions on the order of use of Prolog clauses in a Dislog clause. We say that an instance of a clause r_i precedes an instance of a clause r_j in a proof tree if either r_i appears in that proof tree to the left of r_j or if r_i dominates r_j. Notice that this notion of precedence is independent of the strategy used to build the proof tree. In the following diagram, the clause: a :- a1 precedes the clause b :- b1. :



To model this notion of precedence, we add to Dislog clauses the traditional linear precedence restriction notation, with the meaning given above:

a < b means that the clause with head a precedes the clause with head b (clause numbers can also be used). When the clause order in a Dislog clause is complete, we use the more convenient notation:

f1 / f2 / .... / fn .

which means that f1 precedes f2 which precedes f3 etc... The relation / is viewed as an accessibility relation.

Another improvement to Dislog clauses is the adjunction of modalities. We want to allow Prolog clauses in a Dislog clause to be used several times. This permits to deal, for example, with parasitic gaps and with pronominal references. We use the modality m applied on a rule to express that this clause can be used any number of times in a Dislog clause. For example, in:

{ f1 , f2 , m(f3) }.

the clause f3 can be used any number of times, provided that f1 and f2 are used. Substitutions for identical variables remain the same as before.

Another notational improvement is the use of the semi-colon ';' with a similar meaning as in Prolog to factor out rules having similar parts:

{ a, b }. and *{ a, c }*

can be factored out as:

{ a, (b ; c) }.

which means that a must be used with either b or c.

3.3 Programming in Dislog

Here is a short and simple example where Dislog turns out to be very well-adapted.

In a conventional programming language, there are several one-to-one or one-to-many relations between non-contiguous instructions. For instance, there is a relation between a procedure and its corresponding calls and another relation between a label declaration and its corresponding branching instructions. Dislog rule format is very well adapted to express those relations, permitting variables to be shared between several definite clause in a Dislog clause. These variables can percolate, for example, addresses of entry points.

We now consider the compiler given in (Sterling and Shapiro 86) which transforms a program written in a simplified version of Pascal into a set of basic instructions (built in the argument). This small compiler can be augmented with two Dislog rules:

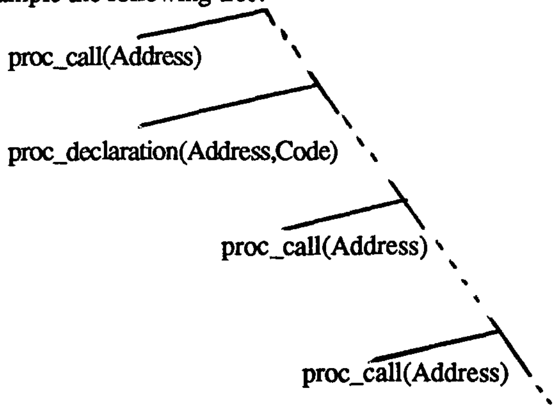
{ procedure declaration, procedure call(s) }.

{ label statement, branching instruction(s) to label}.

In order for a procedure call to be allowed to appear before the declaration of the corresponding procedure we do not state any linear precedence restriction. Furthemore, procedure call and branching instruction description rules are in a many-to-one relation with respectively the procedure declaration and the label declaration. A procedure call may indeed appear several times in the body of a program (this is precisely the role of a procedure in fact). Thus, we have to use the modality m as follows:

{ procedure declaration, m(procedure call) }.
{ label statement, m(branching instruction to label) }.

In a parse tree corresponding to the syntactic analysis of a Pascal program, we could have, for example the following tree:



The main calls and the Dislog rules are the following:

```
parse(Structure) --> [program],
    identifier(X),   [';'],
    statement(Structure).
statement((S;Sa) --> [begin],
    statement(S),
    rest_statement(Sa).
statement(assign(X,V)) -->
    identifier(X),   [':='],
    expression(V).

/* procedure declaration and call */
{ (statement(proc_decl(N,S)) -->
    [procedure],
    identifier(N),
    statement(S),
    [end] ) ,
    m( statement(proc_call(N,S)) -->
        identifier(N) ) }.
/* label declaration and branching */
{ (statement(label(N)) -->
    identifier(N),
    [':'] ) ,
    m( statement(goto(N)) -->
        identifier(N) ) }.
```

We have carried out an efficient and complete implementation for Dislog rules which are compiled into Prolog clauses.

4. Expressing movement rules in Dislog

A way of thinking to move-α (as in Sells 85) is that it expresses the 'movement' part of a relation between two structures. We quote the term movement because, in our approach, we no longer deal with d-structure and no longer have, thus, movements but rather long-distance relations or constraints.

We think that, in fact, move-α is itself the relation (or prototype of relation) and that the constraints (case assignment, θ-marking, bounding theory, etc...) are just specific arguments or constraints on that relation: everything is possible (relation) and constraints filter out incorrect configurations. From this point of view, Dislog is a simple and direct computational model for move-α.

4.1 Expressing movement in Dislog

The relativisation rule given above is expressed in a straightforward way by a Dislog clause. That Dislog clause is composed of two Prolog(-like) clauses. The first clause deals with the adjunction of the N(+Pro) to the COMP and the second clause deals with the derivation of the N2 into a trace. A shared variable I permits to establish the co-indexation link. The Dislog clause is the following, in which we adopt the X-bar syntax terminology:

```
    xp(comp,0,_,_,_) --> xp(n,0,pro(Case),I,_) ,
xp(comp,0,_,_,_). /
        xp(n,2,Case,I,_) --> trace(I).
```

An xp is a predicate which represents any category. The category is specified in the first argument, the bar level in the second, syntactic features in the third one (oversimplified here), the fourth argument is the co-indexation link and the last one, not dealt with here, contains the logical form associated with the rule. Notice that using identical variables (namely here I and Case) in two different clauses in a Dislog clauses permits to transfer feature values in a very simple and transparent way.

The passive construction is expressed in a similar way. Notice that we are only interested in the s-structure description since we produce annotated surface forms (from which we then derive a semantic representation). The passive construction rule in Dislog is:

```
    xp(infl,1,_,_,_) --> xp(n,0,_,I,_), xp(infl,1,_,_,_)
/ xp(n,2,_,I,_) --> trace(I).
```

Case and θ-role are lexically induced. Following a specification format like in (Sells 85), we have, for example, for the verb *to eat*, the following lexical entry:
eat, V, (subject:NP, agent), (object:NP, patient), assigns no case to object.
which becomes with the passive inflection:
eaten, V, (object: NP, patient), assigns no case.
(the by-complement is also lexically induced by a lexical transformation of the same kind with: iobject:NP, agent, case: ablative)

Let us now consider the subject-to-subject raising operation. At d-structure, the derivation of an N2 into the dummy pronoun *it* is replaced by the derivation of that N2 into an overt noun phrase. This is formulated as follows in Dislog:

```
    xp(infl,2,Case,_,_)  -->  xp(n,2,Case,I,_),
xp(infl,1,_,_,_)  /
    xp(n,2,Case,I,_) --> trace(I).
```

In order to properly anchor the N2, we have to repeat in the Dislog rule a rule from the base component (rule with infl). Once again, this is lexically induced from the description of the verb *to seem*: when the N2 is raised, the proposition following the completive verb has no subject, it is tenseless, i.e. in the infinitive form. Finally, notice the case variable, designed to maintain the case chain.

The wh-construal construction at LF is dealt with in exactly the same manner, an N2(+pro) is adjoined to a COMP node:

```
    xp(comp,2,_,_,_)  -->  xp(n,2,pro(Case),I,_),
xp(comp,2,_,_,_)  /
    xp(n,2,Case,I,_) --> trace(I).
```

Case permits the distinction between different pronouns. Notice that this rule is exactly similar to the relative construction rule.

Dislog rules describing movements can be used in any order and are independent of the parsing strategy. They are simple, but their interactions can become quite complex. However, the high level of declarativity of Dislog permits us to control movements in a sound way.

The movement construction rules given abov have many similarities. They can be informally pı together to form a single, partially instanciate movement rule, roughly as follows:

```
    (   (xp(infl,1,_,_,_)   -->   xp(n,0,_,I,_
xp(infl,1,_,_,_) ) ;
    (xp(infl,2,Case,_,_)  -->  xp(n,2,Case,I,_
xp(infl,1,_,_,_) ) ; etc....   /
    xp(n,2,(Case;pro(Case)),I,_) --> trace(I) ).
```

4.2 Other uses of Dislog fo natural language processing

Dislog has many other uses in natural languaş processing. At the semantic level, it can be used in convenient way as a computational model to deal wi quantifier raising, with negation and modality operat raising operations or to model some meanii postulates in Montague semantics. Dislog can al: provide a simple model for temporal relatioı involving the notion of (partial) precedence of actioı or events.

Semantic interpretation or formula optimisati often involves putting together or rewriting elemer which are not necessarily contiguous in a formul Dislog rules can then be used as rewriting rules.

Finally, at the level of syntax, we have shown (Saint-Dizier 87) that Dislog can be efficiently used deal with free phrase order or free word orc languages, producing as a result a normaliz syntactic tree. Dislog can also be used to skip parts sentences which cannot be parsed.

4.3 Formal grammatical aspects Dislog rules

A Dislog rule can be interpreted by a **ter attribute grammar**. A term attribute grammar h arguments which are terms. It is a context-fr grammar that has been augmented with conditions (arguments) enabling non-context-free aspects oi language to be specified. A Dislog rule can translated as follows into a term attribute gramm Consider the rule:
a --> b / c --> d.
a possible (and simple) interpretation is:
a(X,Y) --> b(X,X1), add(X1,[c-->d],Y).
b(X,Y) --> withdraw([c-->d],X,Y1), d(Y1,Y).
When a-->b is executed, the rule c-->d is stored an argument (X and Y represent input and outı arguments for storing these rules to be executed, l strings of words are stored in DCGs). c-->d can oı

be executed if it is present in the list. At the end of the parsing process, the list of rules to be executed must be empty (except for rules marked with modality m). Notice also that shared variables in a Dislog rule are unified and further percolated when rules are stored by the procedure add.

Dislog rules can be used to express context-sensitive languages. For example, consider the language L= {$a^n b^m c^n d^m$, n, m positive integers), it is recognized by the following grammar:

S --> A, B, C, D.
A --> [a], A / C --> [c], C.
B --> [b], B / D --> [d], D.
A --> [a]. B --> [b].
C --> [c]. D --> [d].

If a, b, c and d are mixed, with the only condition that the number of a's is equal to the number of c's and the number of b's is equal to the number of d's, we have:

{ (S --> [a], S), (S --> [c], S) }.
{ (S --> [b], S), (S --> [d], S) }.
S --> [a] / [b] / [c] / [d].

Bounding nodes and modalities can also be added to deal with more complex languages.

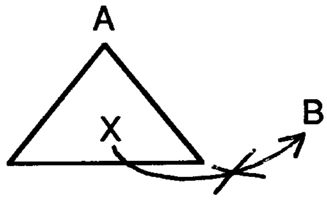
4.4 Related works

Dislog originates a new type of logic-based grammar that we call **Contextual Discontinuous Grammars.** The closest formalisms to Dislog are Extraposition Grammars (Pereira 1981) and Gapping Grammars (Dahl and Abramson 1984). As opposed to Gapping Grammars, Dislog permits to deal with trees rather than with graphs. Gapping Grammars are of type-0 and are much more difficult to write and to control the power of. Compared to Extraposition Grammars, Dislog no longer operates movements of strings and it is also more general since a Dislog clause can contain any number of Prolog clauses which can be used in any orderand at any place within a domain. Extraposition grammars also involve graphs (although much simpler than for Gapping Grammars) instead of trees, which are closer to the linguistic reality. The implementation of Dislog is about as efficient as the very insightful implementation provided by F. Pereira.

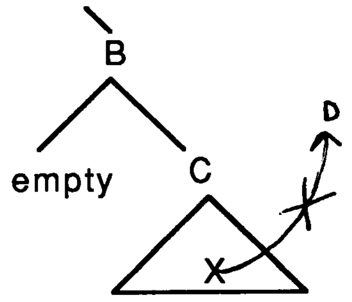
More recently (Dahl, forthcoming), Static Discontinuity Grammars have been introduced, motivated by the need to model GB theory for sentence generation. They permit to overcome some drawbacks of Gapping Grammars by prohibiting movements of constituents in rules. They have also borrowed several aspects to Dislog (like bounding nodes and its procedural interpretation). Dislog is however more general and more powerful because it deals with unordered sets of rules rather than with a single, rigid rewriting rule, it also permits to introduce modalities and no extrasymbols (to represents skips or to avoid loops) need to be introduced (see Saint-Dizier 88b).

5. Bounding Theory in Dislog

Bounding theory is a general phenomena common to several linguistic theories and expressed in very similar ways. Roughly speaking, Bounding theory states constraints on the way to move constituents, or, in non-transformational terms on the way to establish relations between non-contiguous elements in a sentence. The main type of constraint is expressed in terms of domains over the boundaries of which relations cannot be established. For example, if A is a bounding node (or a subtree which is a sequence of bounding nodes), then the domain of A is the domain it is the root of and no constituent X inside that domain can have relations with a constituent outside it (at least not directly):



or, if A represents a sequence B ... C of bounding nodes:



In Dislog, if an instance of a Dislog clause is activated within the domain of a bounding node, then, the whole Dislog clause has to be used within that domain. For a given application, bounding nodes are specified as a small database of Prolog facts and are interpreted by the Dislog system.

In the case of Quantifier Raising, we have several types of bounding nodes: the nodes of syntax, nodes corresponding to conjunctions, modals, some temporal expressions, etc... Those nodes are declared as bounding nodes and are then processed by Dislog in a way transparent to the grammar writer.

6. An implementation of Dislog for

natural language processing

We have carried out an specific implementation of Dislog for natural language parsing described in (St-Dizier, Toussaint, Delaunay and Sebillot 1989). The very regular format of the grammar rules (X-bar syntax) permits us to define a specific implementation which, in spite of the high degree of parametrization of the linguistic system, is very efficient.

We use a bottom-up parsing strategy similar to that given in (Pereira and Shieber 1987), with some adaptations due to the very regular rule format of X-bar syntax rules, and a one-step look-ahead mechanism which very efficiently anticipates the rejection of many unappropriate rules. The sentences we have worked on involve several complex constructions; they are parsed in 0.2 to 2 seconds CPU time in Quintus Prolog on a SUN 3.6 workstation.

7. Perspectives

In this paper, we have presented a simple, declarative computational model for movement theory in Government and Binding. For that purpose, we have introduced Dislog, a logic programming language built on top of Prolog, designed to express in a simple, transparent and concise way relations or constraints between non-contiguous constituents in a structure. Although Dislog is still in an early stage of development, it seems a promising language for natural language processing and also to represent and to program several kinds of problems where the idea of non-contiguity is involved. The efficient implementation we have carried out permits to use Dislog on a large scale. We have designed a prototype parser which includes our model of movement rules, the GB base component, a quite extensive lexicon and semantic compositional rules to build logical formulas. We also use the same model for natural language generation.

Acknowledgements

I would like to thank A. Borillo, M. Borillo, C. Brown, V. Dahl, C. Delaunay, G. Lapalme, P. Sebillot and J. Virbel for their comments on ealier versions of this work. This project is supported by the INRIA and by the PRC Communication Homme-machine of the CNRS.

References

Berwick, R. and Weinberg, A., *The Grammatical Basis of Linguistic Performance*, MIT Press, 1986.

Brown, C., Pattabhiraman, T., Massicotte, P., Towards a Theory of Natural Language Generation: the Connection between Syntax and Semantics, in: *Natural Language Understanding and Logic Programming II*, V. Dahl and P. Saint-Dizier Edts, North Holland 1987.

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

Chomsky, N., *Barriers*, Linguistic Inquiry monograph nb. 13, MIT Press 1986.

Colmerauer, A., Metamorphosis Grammars, in: *Natural Language Understanding by Computer*, Lecture notes in Computer Science, L. Bolc Edt., Springer-Verlag, 1978.

Dahl V., Abramson, H., On Gapping Grammars, *Proc. of the 3rd Logic Programming Conference*, Uppsala, 1984.

Farinas del Cerro, L., Arthaud, A., Molog: Programming in Modal Logic, *Fifth Generation Computing journal*, 1985.

May, R., *Logical Form*, Linguistic Inquiry monograph nb. 12, MIT Press, 1986.

Mukai, K., Unification over Complex Indeterminates, *Fifth Generation Computer journal*, 1985.

Nadathur, G., Miller, D., An overview of λ-Prolog, Technical report MS-CIS-88-40, University of Pennsylvania, 1988.

Pereira, F., Logic for Natural Language Analysis, SRI technical report 275, Stanford, 1985.

Pereira, F., Sheiber, S., *Prolog for Natural Language Analysis*, CSLI lecture Notes, Chicago University Press, 1987.

Saint-Dizier, P., Contextual Discontinuous Grammars, in: *Natural Language Understanding and Logic Programming II*, V. Dahl and P. Saint-Dizier Edts, North Holland, 1987.

Saint-Dizier, P., Dislog, Programming in Logic with Discontinuities, *Computational Intelligence*, vol. 5-1, 1988.

Saint-Dizier, P., Foundations of Dislog, programming in Logic with Discontinuities, in *proc. of FGCS'88*, Tokyo, 1988.

Saint-Dizier,P., Toussaint,Y.,Delaunay,C., Sebillot,P., A Natural language processing system based on principles of government and binding theory, in *Logic Programming and Logic grammars*, P. Saint-Dizier and S. Szpakowicz Edts, Ellis Horwood, 1989.

Sells, P., *Lectures on Contemporary Syntactic Theories*, CSLI lecture notes no 3, Chicago University Press, 1985.

Stabler, E., Parsing with Explicit Representations of Syntactic Constraints, in: *Natural Language Understanding and Logic Programming II*, V. Dahl and P. Saint-Dizier Edts, North Holland, 1987.

Sterling, L., Shapiro, S., *The Art of Prolog*, MIT Press, 1986.