Constrained discontinuous grammars. A linguistically motivated tool for processing language

Veronica Dahl, Patrick Saint Dizier

INRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 573

CONSTRAINED DISCONTINUOUS GRAMMARS

A LINGUISTICALLY MOTIVATED TOOL FOR PROCESSING LANGUAGE

Verónica DAHL
Patrick SAINT-DIZIER

Octobre 1986

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 - RENNES CÉDEX
FRANCE
Tél. : (99) 36.20.00
Télex : UNIRISA 95 0473 F

CONSTRAINED DISCONTINUOUS GRAMMARS - A Linguistically Motivated Tool for Processing Language

GRAMMAIRES DISCONTINUES CONTRAINTES - Un outil motivé linguistiquement pour traiter les langues.

Publication Interne n° 309 - Septembre 1986
40 pages

Verónica DAHL
Dept. of Computer Science
Simon Fraser University
BURNABY B.C. V5A 1S6
Canada

Patrick SAINT-DIZIER
I.R.I.S.A.
Campus de Beaulieu
35042 RENNES-Cédex
France

ABSTRACT

We propose a logic grammar formalism that
(a) accepts non-context-free rewriting rules where one can either specify each symbol, or focus on non-contiguous, related symbols while skipping intermediate, anonymous substrings of symbols;
(b) accepts general constraints on rule applicability. These constraints are described modularly and checked dynamically, since they refer to given forms of the parse tree.

Within this formalism, we show that key traits of transformational theory, including filtering through traits, are easily expressed. We provide examples, tested within a concrete implementation.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L.A. 227)
UNIVERSITÉ DE RENNES 1 I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

GRAMMAIRES DISCONTINUES CONTRAINTES - Un Outil Motivé Linguistiquement pour Traiter les Langues.

Veronica DAHL

Dept. of Computer Science
Simon Fraser University
BURNABY B.C. V5A 1S6
CANADA

Patrick SAINT-DIZIER
IRISA
Campus de Beaulieu
35042 RENNES CEDEX
FRANCE

RESUME

Nous proposons un formalisme de grammaire logique qui :
(a) accepte des règles de réécriture non contexte-libre dans lesquelles peuvent apparaître des discontinuités.

(b) accepte des contraintes de filtrage sur l'applicabilité de ces règles en fonction de la forme de l'arbre d'analyse.

Dans ce cadre, nous montrons comment les éléments majeurs de la théorie transformationnelle peuvent être exprimés. Des exemples concrets sont donnés ainsi que des explications détaillées sur l'implémentation que nous avons réalisée.

1_ INTRODUCTION

Since the development of the first logic grammar formalism by A. Colmerauer in 1975 [5], and of the first sizeable application of logic grammars by V. Dahl in 1977 [6], several variants of logic grammars have been proposed. Some of these were motivated by ease of implementation (Definite Clause Grammars, DCG, [20]). Others provided a general treatment of some natural language processing problem such as movement rule simplification (Extraposition grammars, XGs, [21]) or coordination (Modifier Structure Grammars, MSGs, [7]). Still others were devised to automate some part of the grammar writing process, such as the automatic construction of parse trees and internal representations (MSG's, opcit, Definite Clause Translation Grammars [1]). Generality and expressive power seem to have been the main concerns underlying all these efforts.

Simultaneously, linguistic developments tended towards higher orders of abstraction and generalization. Transformational theory, in particular, has contributed much to precise formalizations of natural language. Basically, a set of core rules is described, and further rules or principles are applied to generate (or recognize) sentences. In recent linguistic developments, economy in the number of rules has been recognized as one of the main objectives, both for the sake of linguistic appropriateness and for the effective transfer of results onto other fields, namely, computational linguistics [2].

Chomsky's efforts to formalize the description of language have inspired the development of many computational models. The idea of having a set of base rules and a set of transformations has been elaborated upon in different ways. Some of these are mostly notational variants of standard theory, in that they move some of the weight from the transformational to the base rule component (cf. Gazdar's derived categories [14], [15]), without achieving much overall economy.

One of the techniques to generalize transformational rules has been the use of metarules: these can be viewed as abstract descriptions that take existing rules of a certain form in order to produce new ones. The production of new rules takes place statically, preceding any actual use of the grammar. Constraints are usually necessary to control the production of new rules, e.g. in view of avoiding cycles [32].

In Chomsky's most recent theory (Government and Binding) [4] [25], overall economy translates into very few base rules plus a single, very general movement rule and a set of principles to discard the ungrammatical sentences that these rules do not filter out. The base rules have a relatively regular format obtained through a high level of generalization that collapses all rules that

previously concerned different phrasal heads (e.g. adjectives, nouns, verbs, etc ...) into a few rules on the abstract category "phrasal head". The unique movement rule also absorbs what were previously several movement rules into one, and any remaining adjustments are achieved through a set of general principles, described in a modular fashion.

In the same spirit of conciseness and modularity, joined with the objective of making linguistic descriptions computationally meaningful, discontinuous grammars were devised in 1981 by V. Dahl [8] [9]. In earlier literature they were misnamed "gapping grammars".[1]

A discontinuous grammar rule allows us to concentrate on those constituents that are relevant to the phenomenon it describes, and to skip over intermediate, unspecified substrings. These substrings are left unanalysed by the rule, but are possibly moved (or duplicated, or deleted) into other positions by the rule's application, for later analysis by other rules.

Discontinuous grammars (DGs) are a generalization of extraposition grammars, in the sense that they allow for right as well as left extaposition, and are not rigit with respect of what becomes of skipped substings (where they should be moved to, whether they can be deleted, duplicated, etc.), or to particular nesting constraints.

For instance, the DG rule:

(1) a, skip(X), b, skip(Y), c --> skip(Y), c, b, skip(X).

can be applied successfully to either of the following strings:

a,e,f,b,d,c

with skips X=e,f and Y= d, and

a,b,d,e,f,c

with skips X= [] and Y= d,e,f.

Application of the rule yields respectively:

d c b e f

[1] "gapping grammar" was a misnomer because our meaning of the word "gap" (now called skip) is quite different from a linguist's. Our gaps are simply a shorthand for a sequence of symbols, whereas in linguistics, a gap refers to the trace left by a moved or a deleted constituent.

and:

d e f c b

We can therefore think of the above DG rule as a shorthand for, among others, the two rules:

(1a) a,e,f,b,d,c –> d,c,b,e,f

(1b) a,b,d,e,f,c –> d,e,f,c,b

If we instead had the rule:

a, skip(X), b, skip(Y), c –> skip(X), c, b, skip(X).

the first string would become:

e f c b e f

Notice that discontinuous grammar rules can be viewed as shorthand for several ordinary (logic grammar) rules, as the above example shows. In this sense they can be considered as metarules. But they are metarules in that their precise instances are constructed dynamically, as they are needed during the parse of a given sentence.

The formalism presented here is an extension of the DG one, to which constraints are added, as we shall see later. These constraints are described separately from the rules, and act like demons that block a rule's application when it attempts a forbidden movement. "Forbidden movement" is described in terms of the shape of a parse tree or graph, and is therefore checked dynamically.

Notice that ordinary logic grammar rules could already be considered as metarules, since they stand for all instances obtained by substituting terms for the variables that appear in the arguments of grammar symbols. DGs generalize this idea, by also allowing variables to stand for finite strings of grammar symbols.

Thus, DGs provide a great degree of economy, in two senses: not only we do not routinely produce specific rules like (1a) and (1b) that might never be used (and that may be infinite in number!), but the use of skips allows us a high degree of generality by which we can collapse what should be several rules in ordinary grammars into one.

Moreover, our metarules follow the same notation as the ordinary logic grammar rules, with the only addition of the "skip" symbol, whereas in other approaches, metarules have their own, ad-hoc representation. Thus, we can

consider ordinary rules as special cases of the general discontinuous rule format, in which there are no skips. In fact, DGs subsume most of the existing logic grammar formalisms as special cases.

It should also be noticed that there seems to be some psychological reality to the idea of skipping intermediate strings. Often, while analysing a sentence, we focus on the next relevant substring, leaving an intermediate one suspended in the background of consciousness, to be brought back into focus later, possibly repositioned with other more closely related substrings.

We can now define Discontinuous grammars more formally. Let F be a set of functional symbols, including the unary symbol "skip", and V an enumerable set of variables. Let $\hat{H}$ be the set of terms constructibles from F and V, i.e., the set of formulas consisting of either a variable, a constant (i.e., a functional expression of 0-arity) and an expression $f(t_1, \ldots, t_n)$, where f is a functional symbol of arity n and the t_i are terms. Let H be the set of ground terms (i.e., terms containing no variables) in $\hat{H}$. Let $V_T \subset \hat{H}$, called the terminal vocabulary. Let $V_N \subset \hat{H}$, called the non-terminal vocabulary, $V_T \cap V_N = \emptyset$. Let $\Gamma \subset V_N$, called the set of skip symbols, be a set of the form: $\{skip(G_1), skip(G_2), \ldots, skip(G_n)\}$, where the G_i are variables. Let $V_S \subset V_N$ be the set of start symbols. Let P be a set of production (meta)-rules of the form:

$$nt, a_0, skip(G_1), a_n, \ldots, skip(G_n), a_n \longrightarrow b_0, skip(G_{i_1}), b_1, \ldots, skip(G_{i_m}), b_m$$

where $m, n \geq 0; nt \in V_N$; $a_i, b_i \in (V_N \cup V_T)^*; skip(G_i) \in \Gamma$, and for all $j, 1 \leq i_j \leq n$. Let ==> be a rewriting relation on $(V_N \cup V_T)^*$, defined as follows:

u ==> v <=> there exists a production rule:

$$a_0 skip(G_1), a_1, \ldots, skip(G_n), a_n \rightarrow$$
$$b_0 skip(G_{i_1}), b_1, \ldots, skip(G_{i_m}), b_m$$

and some substitution θ of terms for variables, such that:

$$u\theta = (a_0 \gamma_1 a_1 \cdots \gamma_{n-1} a_n)\theta$$

for some $\gamma \in (V_N \cup V_T)^*$

$$v = (b_0 \gamma_{i_1} b_1 \cdots \gamma_{i_m} b_m)\theta$$

where for all $j, 1 \leq i_j \leq n$.

The quintuple $G = (V_N, V_T, \Gamma, V_S, P)$ is called a *discontinuous grammar*.

We now define $\stackrel{*}{==>}$ to be the reflexive, transitive closure of $==>$. The language generated by G is:

$$L(G) = \{t \in V^* : S \in V_S \text{ with } S \stackrel{*}{==>} t\}$$

Notice that each of the metarules in P stand for all instances of it --i.e., for all rules obtained from the metarule by substituting terms for variables and skips for matching substrings in a parsing state.

Section 2 presents several uses of DGs. Section 3 introduces some ideas on the way to write DG rules for linguistic purposes and section 4 gives several properties of DGs, some of which are shared by other logic based grammar formalisms. Finally, section 5 shows how DG rules can be constrained in order to prevent incorrect derivations. Section 2 overlaps somewhat with material previously presented in not easily accessible publications [9,8].

2_ SOME EXAMPLES -- THINKING IN TERMS OF GAPS

In this section, we examine several natural as well as artificial language processing problems in terms of discontinuous grammar rules.

2.1_ Coordination

The following sample grammar handles sentence coordination and reconstitutes the meaning representation of an elided object.

```
sentence(and(S1,S2)) -->
  sent(S1), and, sent(S2).

sentence(S) --> name(K),
  verb(K,P,S), object(P).

object(P) --> determiner(X,P1,P)
    noun(X,P1).
object(P), and, skip(G), object(P)
 --> [and], skip(G), object(P).

determiner(X,P,the(X,P)) --> [the].
```

```
noun(X,train(X)) --> [train].

name(mary) --> [mary].
name(john) --> [john].

verb(X,Y,saw(X,Y)) --> [saw].
verb(X,Y,heard(X,Y)) --> [heard].
```

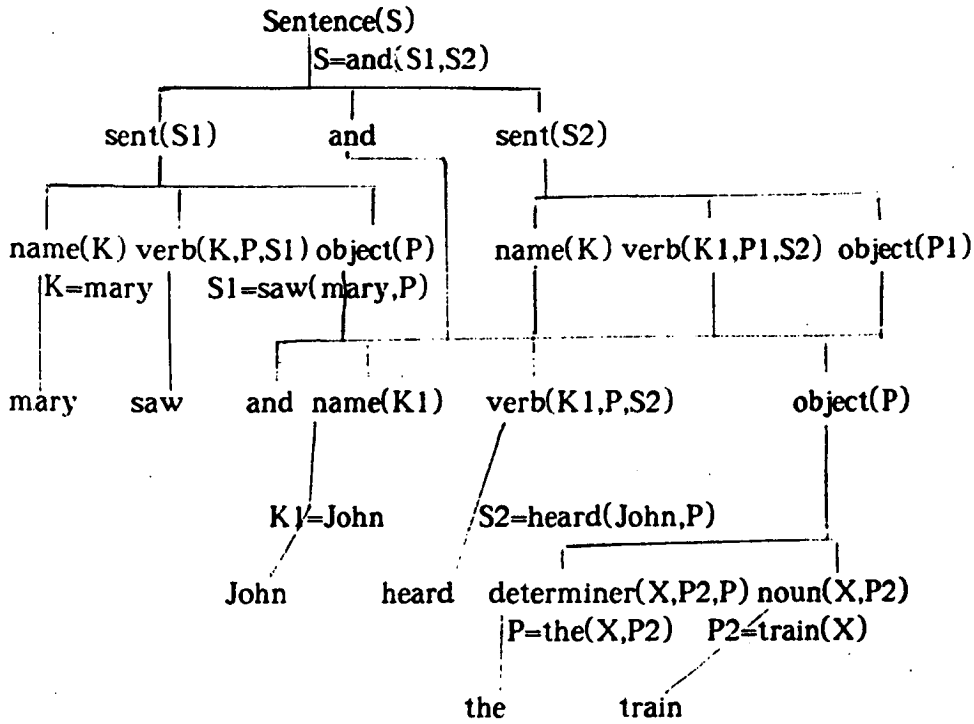
As in Metamorphosis grammars, a normalizing rule of the form nt-->[nt]. is also necessary for each non-terminal, non-initial symbol nt contained in the left hand side of a rule. As these rules can be constructed automatically by a grammar preprocessor, we shall consider them transparent to the grammar writer and disregard them in all that follows.

The second rule for 'object' elides an expected object followed by a skip and reconstructs its internal representation 'P' through unification with the meaning representation of the object in the second sub-sentence. The skip in between 'and' and the object is recopied unanalysed.

A derivation graph for the sentence 'mary saw and john heard the train', shown in Fig. 1, might help visualize the working of the grammar.

We have labelled each rule application with the substitutions used, and circled the skip. The final value obtained is:

```
S=and(saw(mary,the(X,train(X))),heard(john,the(X,train(X))))
```



2.2_ Relativization

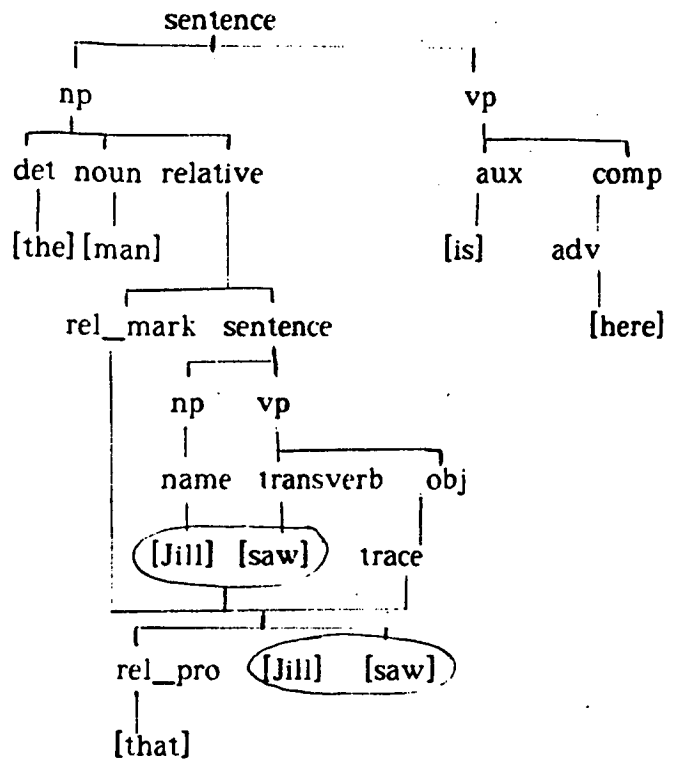
Relative clauses often can be viewed as more canonical sentences whose parts have undergone changes and movement. Thus, a sentence as:
"The man that Jill saw is here."
can be viewed as the result of transforming:
The man [Jill saw the man] is here
by moving the second occurrence of "the man" to the beginning of the embedded clause and replacing it by a pronoun. The following grammar parses sentences such as the above into logical structures such as:

```
the(X,and(man(X),saw(jill,X)),here(X))
```

```
sentence(P) --> np(X,P1,P), vp(X,P1).

np(X,P1,P) --> det(X,P2,P1,P), noun(X,P3), relative(X,P3,P2).
np(X,P,P) --> name(X).

vp(X,P) --> trans-verb(X,Y,P1), object(Y,P1,P).
vp(X,P) --> aux(be), comp(X,P1,P).
```

```
relative(X,P1,and(P1,P2)) --> rel-marker(X), sentence(P2).
relative(X,P,P) --> [].

rel-marker(X), skip(G), trace(X,P1,P) --> rel-pronoun, skip(G).

object(X,P,Q) --> np(X,P,Q).
object(X,P,P) --> trace(X).

comp(X,P,P) --> adverb(X,P).


noun(X,man(X)) --> [man].

aux(be) --> [is].

adverb(X,here(X)) --> [here].

det(X,P1,P2,the(X,P1,P2)) --> [the].

rel-pronoun --> [that].

name(jill) --> [jill].

trans-verb(X,Y,saw(X,Y)) --> [saw].
```

The technique used is to rewrite a relative clause into a marker followed by a (canonical) sentence. The marker is then used, together with a trace left for the elided object, to perform the contextual changes mentioned. Graphically, we can view the rewriting as follows, leaving arguments out for clarity:



Similarly, Halliday's example on discontinuous nominal groups [16] :

(S1) The man from the Gaz Board came.

can become:

(S2) The man came from the Gaz Board.

through the DG rule:

prep-phrase. skip(X) --> skip(X), prep-phrase.

For simplicity, we have not included here any arguments other than the one representing the skip.

Notice that, while sentence (S1) is not ambiguous, (S2) has the two meanings derived from attaching the prepositional phrase to either the noun phrase or the verb. The above DG rule does not destroy any of these meanings, since it only concerns the attachment to the noun phrase. Other grammar rules will generate the case where the complement modifies the verb.

Of course, this DG rule is actually too general: it allows to move any

prepositional phrase, whether it modifies the head of the subject noun phrase or not. In section 5, we shall see how to deal with this kind of problem.

2.3_ Right Extraposition

In natural language, some movement phenomena are more naturally viewed as right rather than left-extraposition, although they could perhaps be forced into left-extraposing formulations.

We next augment the little grammar shown in 2.2 by adding two rules for extraposing the whole relative clause to the right. Both the left and the right extraposing rules can interact in the same sentence, as the example shows. The added rules are:

```
relative(X,P1,P), skip(G) --> skip(G), rightex(X,P1,P).

rightex(X,P1,and(P1,2)) --> rel-marker(X), sentence(P2).
```

(One of these rules, as we shall see in section 4, is needed for avoiding loops)

The grammar now parses the sentence:
"The man is here that Jill saw."
into the same representation as its paraphrase.

2.4_ Interactions between different discontinuous rules

Consider the language $L(G) = \{a^n\ b^m c^n\ d^m : n, m \in N\}$, which can be described by the following DG:

```
s --> as, bs, cs, ds.

as -->[].
as, skip(X), xc --> [a], as, skip(X).

bs --> [].
bs, skip(X), xd --> [b], bs, skip(X).

cs --> [].
cs --> xc, [c], cs.

ds --> [].
ds --> xd, [d], ds.
```

This is a perfectly good DG. XGs cannot, however, be used in this

situation because of the XG constraint on the nesting of skips: two skips must either be independent, or one skip must lie entirely within the other.

2.5_ Avoiding artifices through straightforward uses of skips

The grammar in 2.4 uses marker symbols "xc" and "xd", whose only function is to leave traces of right-extraposed a's and b's, in place where they can easily be evened out with c's and d's, respectively. However, thinking in terms of markers is in many cases simply a residue from the writing styles imposed by less powerful grammars.

Similarly, the language $L(G) = \{a^n\ b^m c^n\ d^m : n, m \in N\}$, for which a formulation with markers was shown in [8], can be more easily formulated as follows:

s --> as, bs, cs, ds.

as, skip(G), cs --> [a], as, skip(G), [c], cs.
as, skip(G), cs --> skip(G).

bs, skip(G), ds --> [b], bs, skip(G), [d], ds.
bs, skip(G), ds --> skip(G).

The second rule simply evens as and cs with a and c, by skipping any intermediate string as a skip G, which is repositioned after regenerating as and cs in the right hand side. The third rule simply makes as and cs vanish.

2.6_ Rules with more than one skip

Using the same technique as for the last grammar shown in 2.5, we can describe the language $L(G) = \{a^n\ b^n c^n\}$ through:

1) s --> as, bs, cs.

2) as, skip(G1), bs, skip(G2), cs -->
   [a], as, skip(G1), [b], bs, skip(G2), [c], cs.

3) as, skip(G1), bs, skip(G2), cs --> skip(G1), skip(G2).

2.7_ DGs and free word order languages

Many languages exhibit, to some extent, case marking through morphology rather than through position in the sentence. Since, in these languages, altering the position does not make the case dubious, some constituents can freely move from their "orthodox" positions into arbitrary or fairly arbitrary locations, as a matter of style, emphasis, etc...

2.7.1_ Totally free word or constituent order

For instance, the Sanscrit sentence "Ramauh pashyati Seetam" (Ramauh sees Seetam) can also appear as:

pashyati Ramauh Seetam
pashyati Seetam Ramauh
Seetam Ramauh pashyati
Seetam pashyati Ramauh
Ramauh Seetam pashyati

This kind of free order of sister constituents where each retains its integrity is easily handled within discontinuous grammars. More interesting is the case in which even the contents of constituents appear to be scrambled up with elements from other constituents (e.g. as in the Walpiri language). Even in Latin or Greek, phenomena such as discontinuous noun phrases, which would appear as extreme dislocation in prose, are very common in verse (and not unusual even in certain prose genres (e.g. Plato's late work, such as the Laws)). A contrived example for Latin would be:

Puella bona puerum parvum amat. (Good girl loves small boy)

where the noun and adjective in the subject and or object noun phrase may be discontinued, e.g. :

Puella puerum amat bona parvum.

In fact all 5! word permutations are possible, and we certainly do not want to write a separate rule for each possible ordering. In DGs, we can simply write:

```
sentence --> noun-phrase(nom), noun-phrase(acc), verb.

noun-phrase(Case) --> adjective(Case), noun(Case).

noun(Case), skip(G) --> skip(G), [word]
                        {dict(noun(Case),Word)}.
```

```
adjective(Case), skip(G) --> skip(G), [Word],
                    {dict(adjective(Case),Word)}.

verb, skip(G) --> skip(G), [Word], {dict(verb,Word)}.

dict(verb, amat).
dict(noun(acc),puerum).
dict(noun(nom),puella).
dict(adjective(acc),parvum).
dict(adjective(nom),bona).
```

Notice that the number of rules needed grows linearly depending on the number of constituents which can freely move.

Another approach is the augmented phrase structure one (Pullum 1982). In Pullum's formulation, phrase structure (meta) rules only indicate immediate dominance, and are supplemented with linear precedence restrictions to indicate what orderings are allowed. For instance, the metarule:

A --> B, C, D

together with an empty set of linear precedence restrictions, stands for all rules where A rewrites into B, C and D in any order. With the restriction: {D<C}, on the other hand, it represents only the rules:

{A--> BDC, A -->DBC, A--> DCB}.

While this notation is concise and expressive for free or relatively free ordering problems, it becomes costly as more orderings are fixed. Also, since precedence restrictions are attached to the whole set of phrase structure rules, it can only deal with grammars in which any two constituents that have been stated to have an order appear in that order no matter what their origin. Gazdar and Pullum make the hypothesis that grammars of natural language will all possess this property.

DGs, on the other hand, can describe different orders of same constituents coming from different rules quite straightforwardly, so they will be appropriate even if the above mentioned hypothesis turns out to be wrong, or if we want to process formal language grammars that do not satisfy it. They also seem more versatile in being able to deal with both fixed and changing order with no significant change in cost.

2.7.2_ Lexically induced rule format and free word order

Another interesting problem that can be handled in DGs is free order of constituents that may or may not be present, as determined by other constituents. Let us assume that each particular verb requires its own set of constituents. Then we could include an argument in the verb symbol, telling us about its specific requirements. Modulo notation[2], this would look like:

```
sentence --> verb(R), R.

verb(R), skip(G) --> skip(G), [W], {ver(W,R)}.

nominative, skip(G) --> skip(G), [W], {nom(W)}.

accusative, skip(G) --> skip(G), [W], {acc(W)}.

dative, skip(G) --> skip(G), [W], {dat(W)}.

ver(pashyati,[nominative,accusative]).

ver(yachchhati,[nominative,accusative,dative]).

nom(ramahuh).
acc(seetayai).
dat(pushpam).
nom(ramauh).

acc(seetam).
```

This grammar accepts :
Ramauh pashyati seetam.
Ramahuh Seetayai pushpam yachchhati. (Ramahuh gives a flower to Seeta)

The "sentence" rule first generates the fixed order "verb, ... ", where what follows the verb will be known as a result of processing the "verb" symbol. This, in turn, is done through skipping any preceding constituents until the verb (W) is found. Its lexical definition has a subcategorization frame (R) associated to it. By virtue of the "sentence" rule, this frame materializes as part of the derivation tree.

The possibility for a rule to take a format specifically suited to the requirements of given words might be useful in theories such as Government and Binding, in which the argument structures required by certain words are described in the lexicon. The frame requirements may be associated to a word appearing in any position, even preceding the actual materialization of one of the requirements.

[2]Normally, the variable R above will unify with a list of constituents, so it cannot be directly used as a metacall in the first rule. Further processing would be needed, but we represent it here as a metacall for simplicity.

For instance, a nominative constituent required by a verb may well appear before that verb, but discontinuous rules may allow us to process the verb beforehand.

In [23], F. Popowich presents a method to deal with partially free word order. In particular, he shows how to express the immediate dominance/ linear precedence format rules [15] by a set of unrestricted discontinuous grammar rules that contain procedural control.

The possibility to lexically induce rule format associated with free word order language description facilities offered by discontinuous grammars make it possible to describe some properties of languages such as Japanese in a very convenient way since :

- the verb appears at the end of the clause.
- subject and complements are case-marked and they may appear in any order before the verb.

For instance, in Japanese, we can have the two following structures:

"*Tomoko ga sono boosi o kabutte ita.*"
(Tomoko that hat wearing was)
"*Sono boosi o Tomoko ga kabutte ita.*"

3_ THE ART OF WRITING DG RULES

The expressive power of DGs, while making it easy to write concise grammars, could be misused in certain cases. One might, for instance, be tempted to describe too much in just one rule. Not that is bad in itself, but linguistic theories usually require more modularity. Thus, one could think of expressing both np-movement and pied piping in just one rule in trying to embed:
"John entertained me with the author's novels."
into the sentence:
"The author is here."
to obtain:
"The author with whose novels John entertained me is here."
In strict DG terms, the following, linguistically inadequate rule, is possible:

```
np(X), skip(Y), prep(P), det, skip(Z), prep(of), np(X) -->
  np(X), prep(P), [whose], skip(Z), skip(Y).
```

Arriving at good DG formulations is not, generally speaking, a trivial matter. While we do not yet have a foolproof method for guiding the grammar writer, here are some principles we have found useful, and which we moreover assume in the remainder of this paper, at least where natural language applications

are concerned.

(1) A rule describes a basic movement operation between constituents, i.e., an operation that cannot, without any additional constraint on application, be decomposed into simpler ones which may act independently from each other.

(2) Only the constituents that play a role in the phenomenon which is described by the rule are explicitly present in it.

(3) A grammar writer usually has precise ideas on what elements are moving, although for a machine examining only the two sides of a rule, it is a priori undecidable. For reasons we shall see later, we require the user to indicate, in an additional argument, which are the elements seen as moving ones in a given rule.

(4) In general, for linguistic applications, skips should not be destroyed since they may have semantic content. Only in those cases where the content is not lost (e.g. when another copy remains) is it safe to delete skips. This is induced by what in linguistics is called the recoverability condition [25].

(5) It is best to avoid moving skips since they can stand for various structures or parts of them and this means that we could be moving several constituents at the same time, thus violating what in linguistics is known as the unit movement constituent.

4_ SOME PROPERTIES OF DGs

Some of the properties of DGs that are worth mentioning are inherited from the logic programming mechanisms to which they are related, and shared by other logic based grammar formalisms:

- **Very high level descriptive ability.** This results in mostly declarative formulations which can be then endowed with procedural meaning transparently to the user.

- **Uniformity of formalisation for different stages of natural language systems .** Logic can serve as the basis for syntactic analysis, for semantic representation and its construction, for formula evaluation, and for representing knowledge bases, as well as for the programming language itself [10], [11], [12], [29].

- **Turing machine power.** As other logic grammar formalisms, DGs have Turing machine power. A DG rule can be transformed into a

recursively enumerable set of DCG rules, since a skip stands for a finite string of grammar symbols, and only a finite number of skips may appear in a rule. Therefore, DGs implemented in Prolog are computationally tractable, as shown in [8]

- **Ease of implementation.** Although DGs are a formalism in their own right, independently of any computer implementation of them, one of their main interests is that it is very easy to compile them or interpret them in Prolog.

- **Feature evolution.** Movements described in a DG rule might induce morphosyntactic feature changes in some rule constituents [28]. If all morphosyntactic features are represented as symbol arguments, feature modifications can be straightforwardly expressed without adding any abstract derivational level [17].

Besides these nice properties which DGs share with other logic based grammars, and which are sometimes more crucial for DGs than for simpler formalisms, DGs have their own advantages:

- **Expressive power.** Although the theoretical power is the same as in other logic based grammars (namely, Turing machine power), DGs allow to express more abstract linguistic phenomena in terms of rules. This, as we have seen, results in more concise formulations. Moreover, as shall be seen next, this degree of generality lends itself better to straightforward expressions of linguistic principles.

- **Modularity.** Constraints can be expressed in other logic based grammars, either through Prolog calls, or through additional arguments. This is, however, too limited, since some of the constraints involve a derivation graph rather than just the current information available to a given rule. Also, amalgamating constraint expression together with rewrite information restricts lisibility and change (as all departures from modularity do). Moreover, this mode of expression encourages ad-hoc, purely intuitive solutions, rather than setting well-founded guidelines.

In Modifier Structure Grammars, the need for examining the derivation history was recognized and dealt with modularly. However, partly because of the complexity of the particular problem attacked (coordination), this was done in a transparent way, over which the user

had little control. DGs, instead, provide a means for the user to express any constraint on rule application that involves a derivational context (cf. Section 5).

Modularity is also an important concern regarding implementation. In systems such as DGs, where constraints are clearly separated from the rules, it is even possible to foresee parallel or distributed implementations of at least some parts of the parsing. Work in this direction is currently being done at Maryland [18].

- **Avoiding loops.** Loops related to movement phenomena can be avoided in DGs (as in other logic based grammars) through symbol renaming. For instance, in the grammar shown in Section 2.3, the symbol "rightex" has been introduced to avoid the loop that would have resulted from the rule:

relative(X,P1,P), skip(G) --> skip(G), relative(X,P1,P).

Although effective, this approach results in adding artificial symbols. For DGs, we can allow rules with elementary loops and make their transformation into loopless rules automatic [23]. In other approaches, symbol proliferation is a part of the formalism itself and can grow undesirably numerous (e.g. Gazdar's derived categories).

5_ CONSTRAINING THE APPLICATION OF DG RULES

As was seen in Section 3, it is easy to underestimate the expressive power of DGs with the risk of overgenerating (i.e. accepting ungrammatical as well as grammatical sentences).

Borrowing from linguistic theories, we can abstract three basic ways of expressing constraints:

(1) by applying filters on the form of the syntactic tree(s) of a sentence,

(2) by adding syntactic and semantic subcategorization features to symbols in rules, and

(3) by controlling the coherence and well-formedness of the logical formula produced, in a way similar to [27].

The first and the last methods are certainly more general than the second one, but the second one is appropriate for expressing constraints from lexically induced information. However, it is not totally clear how much the three methods overlap or complement each other.

In this section we deal with methods (1) and (2), since they are closer to the idea behind discontinuous grammars, and directly implementable in them. Moreover, we do not want to establish, a priori, logical formulas as the only means of sentence representation.

Furthermore, although control of movements in Phrase Structure syntax and X-Bar syntax are different systems than feature-based syntax ones, they are, in practice, very often associated. Indeed, a higher level of generality (and explanatory adequacy) could be attained in control of movements if we can express the fact that, for instance, NP and PP form a natural distributional class.

In this work, we take Ross's constraints --one of the most evolved set of constraints developed in transformational theory-- as a first stage in our study of constraint expression in DGs [26]. Our ultimate goal is to show how to incorporate into DGs the principles of Government and Binding Theory, which appear to subsume, together with a very general movement rule, many of the movement phenomena described otherwise in the literature. To our knowledge, there is very little existing work on implementing Government and Binding theory through logic grammars [31], [33]. However, these efforts usually simulate movement through grammar arguments, and remain in the context-free spirit.

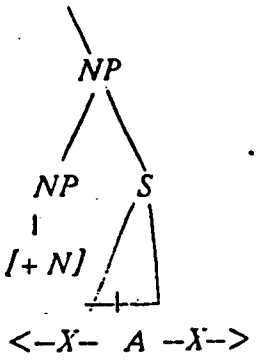
Next we take some examples within the theory of control and complementation [3] using a complex feature system, and we show how to express them through DGs, in a way which prevents incorrect movements.

5.1_ Syntactic tree filtering

This section introduces two of Ross' most representative constraints: the complex NP constraint and the sentential subject constraint, and shows how to express them within DGs.

(a) The complex NP constraint :
"No element contained in a sentence dominated by a noun phrase with a lexical head noun may be moved out of that NP by transformation. "

To put it graphically, this constraint prevents any constituent A from moving out of S:



This constraint allows the transformation of the sentence:

" I believed that Otto was wearing a hat. " into:

"The hat which I believed that Otto was wearing. "

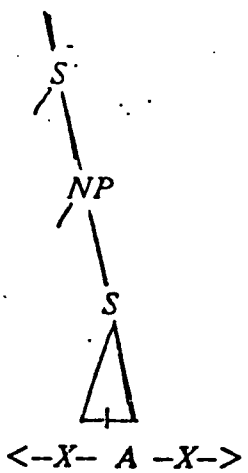
But it blocks the transformation of:

"I believed [[*the claim*]$_{NP}$ [*that Otto was wearing a hat*]$_{S}$]" into:

* "The hat which I believed the claim that Otto was wearing. "

(b) The sentential subject constraint :
"No element dominated by an S may be moved out of that S if that node S is dominated by an NP which is itself immediately dominated by an S. "

This means that A cannot move out of the lowest S in the diagram :



More formally, a Ross' constraint ci can be represented by the structure:

$Ci\,(R_i\,.\,N_i)$,

where:
- R_i is a path in a tree that describes the context of the constraint. We say path in a tree because only relevant branches are indicated.
- N_i is the node that dominates the domain out of which no element can move. In the example below, the variable Z associated to N_i is used as a bench-mark to unambiguously locate N_i in R_i .

For instance, the complex NP constraint is represented by:

ci(np(np(N),s(Z)),s(Z)).

and the sentential subject constraint, by :

cj(s(np(s(Z))),s(Z)).

The notation: A(B) is used to indicate that A does not necessarily immediately dominate B, B has only to be a descendant of A.

Constraints are defined in a separate module than that of rules. The DG processor, whose implementation details are given in the appendix, will automatically add an appropriate predicate to check constraints, as well as all the other machinery needed (e.g. the construction of the derivational context to which the path R_i will be compared).

Constraints are not only useful for natural languages, but also for artificial ones. For instance, if one wants to change an arithmetic expression like:

(a+b)/c × (f+g)/e

into:

(a+b) × e/c × (f+g)

this is possible because the root node of the first divisor's syntactic tree is a product. We can express this simplification rule through a DG rule such as:

exp(X), div, exp(Y),div, elem(Z) --> [(], exp(X), [)], prod, elem(Z), div, exp(Y).

We can express a constraint to block application of this rule if the root node of the syntactic tree of exp(Y) is not a multiplication sign. This constraint could look like:

C1(div(add(E1),prod(E2)),prod(E2))

5.2_ USING FEATURES TO CONTROL LONG DISTANCE DEPENDENCIES.

In this section, we show how features can be appropriately used to control long distance dependencies and relations. As an illustration, we study here the control on complementation by open clauses, but the formalism is also directly applicable to other operations such as: raising, passivization, dummy it-extraposition or there insertion. The use of features to express constraints is very convenient since it permits elegance and conciseness without introducing excessive extra-power into the system. This approach is also adopted in LFG [3] and the examples presented here owe much to J. Bresnan's work. Features, expressed by logical variables, have the form of terms. Thus, they may encode complex feature systems [28]. They automatically percolate in other rules via unification. In addition, feature specification is viewed in the spirit of a declarative language, which makes the expression of constraints easier.

Control introduced on complementation by open clauses is a relation of referential dependence between an unexpressed subject of an open clause and an expressed or unexpressed constituent of another clause that dominates the former. Some of the properties of the unexpressed subject that are referentially determined are those of the constituent that controls it. The grammatical function associated with an open clause is either a complementizer (noted xcomp) or an adjunct (noted xadj). xcomp is lexically introduced whereas xadj is introduced at the level of rules.
For example, in the sentence:

"John seems tired to Sue. "

"John" is in a referential relation with "tired", "tired" has the grammatical function xcomp because the verb "to seem" expects an xcomp referentially dependent on its np-subject.
In the sentence:

"Mary entered the office with blue windows."

"with blue windows" is an xadj grammatical function because it is not lexically induced by the verb "to enter".

We express xcomp-complementation in verbs by an additional feature "completive" with attached values subj, obj or obj2, which states which np in the clause referentially controls the xcomp. The correct referential relation for xadj-adjuncts is only controlled by general syntactic and semantic features.

As all structures in a sentence have an argument that indicate their grammatical function, several constraints on extraposition can be directly encoded

by features, such as:

- verbs whose subject is in a referential relation with an xcomp cannot be passivized:

"John intends to give a present to Mary."
* "It was intended by John to give a present to Mary."
The corresponding DG rule (in a somewhat simplified notation) is:

```
np(X) v(completive(not(subj))) skip(Z) np(Y)  —>
    np(Y) v [by] np(X) skip(Z).
```

The argument "completive(not(subj))" states that the value attached to the feature "completive" of the verb must not be equal to "subj", which stands for the grammatical function subject [28].

- elements may, a priori, be extraposed, questioned or relativized, out of xcomp-complementizers but not from xadjs [19]. For example:

"John seems tired to Mary." which may be rewritten into:
"Tired, John seems to Mary."
But: "John fell asleep, bored by chamber music." cannot become
" *What did John fall asleep [bored by]. ?"

This constraint can also be expressed by features, but the most appropriate way to express it is to add a constraint on the value of the function feature, at the level of the grammar preprocessor, for each symbol (1) marked as being moved and (2) that may potentially have the function xadj:

<symbol>(function(not(xadj)))

Similarly, raising operations (a subtree is raised in the syntactic tree) are allowed only under some conditions, which can be expressed through features:
- the raised np must be a subject,
- in its raised position, it has the function: subj, obj or obj2.
For raising movements, features are a necessary complement to tree filtering devices in order to control (1) the grammatical function of the raised element (via features) and (2) where this element moves to (via tree filtering).

A more general example of the relations between features and tree filtering constraints within the transformational framework is the following. Assume that S, S-Bar and NP are bounding nodes in English for the subjacency constraint. This constraint then prevents the extraction of a wh-phrase out of the complement of a verb dominated by an S-Bar. But, clearly, this is not always true. Chomsky's

solution to this problem is to subcategorize some verbs by a feature as permitting a following non-interrogative tensed S-Bar complement that indicates that this S-Bar is not a bounding node.

Finally, the use of logical variables to represent features in logic-based grammars is very convenient to control relations in multiple embedded structures, as in:

"It seems that John is sick." which after three raising operations becomes:

"John is likely to be believed to seem to be sick."
because features percolate in other rules in a way that is completely transparent to the grammar writer.

6_ DISCUSSION AND CONCLUSION.

We have introduced the DG formalism, and shown its convenience with respect to several language processing problems, both of syntactic and semantic natures. We have also shown that DGs can be restricted by tree and feature filtering devices in order to control their power. These two ways of expressing constraints seem to be elementary: the first one deals with general tree configurations whereas the latter deals with more specific problems that include semantic aspects. These constraints are expressed by convenient logic programming techniques: tree matching, PROLOG calls included in rules, constraints and complex feature propagation by logical variables. We have carried out, in PROLOG, a fully implemented version of the preprocessor of discontinuous rules described in this work.

There exist other approaches to restrict the power of DG. In particular, it is possible (1) to assign arguments to skip symbols that contain information about the structures the skip undertakes, and (2) to assign a number to each DG rule so as to define a partial order for the execution of rules. These approaches have been successfully used to express ID/LP by DG [23].

Assigning features to skip symbols can be really convenient at first glance, for example to prevent an adjective of an np-object to be left extraposed. For that purpose, we could write:
det, skip(X(function(subj))), adj ——→ adj, [,], det, skip(X).
This rule means that all the symbols subsumed in skip(X) have to have the grammatical function "subject". However, we believe that this approach is very intuitive from a linguistic point of view, and might lead to very complex feature expressions in skips as soon as skips subsume several different structures or part of structures, that are fairly well defined at this level.

Assigning numbers to each DG rule seems a priori a more reasonable

approach. A rule with number j may be executed after a rule with number i if and only if j>i. However, apart from the fact that this approach is quite low level and induces large duplication of rules, we believe that it is not fully reliable since (1) problems of circularity may arise between two ordered sets of rules linked to two different constraints and (2) problems of infinite reduplication appear within an ordered set of rules as soon as one wants to describe recursively embedded structures.

So far, discontinuous grammars associated with filtering devices appear to be a convenient formalism to deal with complex and constrained linguistic phenomena. Apart from efficiency problems that remain to be solved, a methodology for writing DG rules (and, more generally, type-0 rules) is also strongly necessary. However, our middle-term goal is to normalize DG rules using principles of Government and Binding Theory. Within this framework, there is only one general movement rule, namely Move-α, associated with constraints expressed under the form of principles. Furthermore, rule format obeys the X-Bar syntax, which is very restrictive. As a result, it seems to us that we should have a more efficient system with a well-defined style for writing rules. It should be noticed that work is in progress about implementing Government and Binding. This work has led to a different version of DGs which seems more specifically suited to Government and Binding theory [13]. Allowing higher-level rules with discontinuities, of course, has its price in terms of efficiency. But, on the other hand, such rules become few in number, since each one represents what would be many non-discontinuous rules, and since various features previously covered by rules can now be covered by constraints.

Moreover, although constraint checking can be seen as an overhead with respect to unconstrained DGs, the early blocking they achieve leads to less backtracking, and therefore, better overall efficiency. Ultimately, as pointed to us by H. Abramson, constraints might become a useful alternative to backtracking, leading to more deterministic processors. An eventual parallel processor of constraints would make the implementation itself inexpensive.

This idea is also applicable for plain Horn Clauses. In this case, constraints would apply on proof derivation trees which can, in fact, be seen as equivalent to sentence derivations, since logic grammars are basically syntactic variants of logic programs.

ACKNOWLEDGEMENTS
We are grateful to H. Abramson, M. Boyer, C. Brown, G. Lapalme, D. Massam, T. Pattabhiraman and P. Sebillot for their useful comments on an earlier draft of this

REFERENCES

[1] Abramson, H.. Definite Clause Translation Grammars. *Proceedings IEEE Logic Programming Symposium*,Atlantic City, N.J., 1982.

[2] Berwick, R.C.. and Weinberger, A.S., *The Grammatical Basis of Linguistic Performance*, MIT Press. 1984.

[3] Bresnan. J.. Control and Complementation. in: *The mental representation of grammatical relations*" J. Bresnan Edt; MIT Press. 1982.

[4] Chomsky. N.. *Lectures on Government and Binding*, Foris Pub., Dordrecht, Holland, 1984.

[5] Colmerauer. A.. Metamorphosis Grammars. in: *Natural Language Communications with Computers*. Lecture notes in Computer Science 63, Springer-Verlag. 1978.

[6] Dahl. V.. Translating Spanish into Logic Through Logic, *American Journal of Computational Linguistics* Vol. 13. 149-164, 1981.

[7] Dahl, V., and McCord, M., Treating Coordination in Logic Grammars, *Journal of Computational Linguistics*, Vol. 9 no. 2, 1983.

[8] Dahl. V. and Abramson, H., On Gapping Grammars, *Proceedings of the 2nd logic programming conference* Uppsala Univ., Sweden, 1984.

[9] Dahl. V.. More on Gapping Grammars. *Proc. of Conference on Fifth Generation Computer Systems, Tokyo*, 1984.

[10] Dahl. V., Logic Programming as a Representation of Knowledge, in: *Computer* Vol. 16, no 10, pp. 106-113, 1983.

[11] Dahl, V., On Database Systems Development Through Logic, in: *ACM Transactions on Database Systems* Vol. 7, no 1, 1982.

[12] Dahl, V., Logic-Based Metagrammars for Natural Language Analysis, in: *Translating Natural Language to Logical Form*, L. Bolc Edt., to appear, Springer-Verlag, 1986.

[13] Dahl, V., Gramáticas Discontinuas, una Herramienta computacional con aplicaciones en la teoría de Gobierno y Nexo, *Revista Argentina de Lingüística*, 1986,

(English version to appear).

[14] Gazdar, G., Phrase Structure Grammars, in: *The nature of Syntactic Representation* P. Jacobson and K. Pullum Edts., D. Reidel, 1982.

[15] Gazdar, G. and Pullum, G.K., Generalized Phrase Structrure Grammars: A theoretical Synopsis, technical report, Indiana Univ. Linguistic Club, Bloomington, IN, 1982.

[16] Halliday, M.A.K., *An Introduction to Functional Grammar*, Arnold, London, 1985.

[17] Jacobson, P., Evidence for Gaps. in: P. Jacobson and G.K. Pullum Edts. *The nature of syntactic representation*, D. Reidel, 1982.

[18] Kasif, S., Kohli, M., Minker, J., PRISM: A Parallel Inference System for Problem Solving. TR 1243, Dept. of computer science, Univ. of Maryland, 1983.

[19] McCord, M., Using Slots and Modifiers in Logic Grammars for Natural Language. TR Univ. of Kentucky 69-80. 1981 and also *Artificial Intelligence* , 1982.

[20] Pereira, F. and Warren, D.H., Definite Clause Grammars for Language Analysis, *Artificial Intelligence* Vol. 6, 1980.

[21] Pereira, F., Extraposition Grammars, *American Journal of Computational Linguistics*, vol. 7, 1981.

[22] Pereira, F., Logic for Natural Language Analysis, SRI international technical note no 275, 1983.

[23] Popowich, F., Unrestricted Gapping Grammars, Ms.Sc. Thesis, Simon Fraser University, Canada, 1985, also Proceedings of IJCAI-85.

[24] Pullum, G.K., Free Word Order and Phrase Structure Rules, in: J. Pustejovsky and P. Sells edts., *Proc. of the 12th annual meeting of the North-Eastern linguistic society*, 1982.

[25] Radford, A., *Transformational Syntax*, Cambridge University Press, UK, 1980.

[26] Ross, J.R., Excerpts from constraints on variables in syntax, in: *On Noam Chomsky, critical essays*, Anchor Press, NY, 1974.

[27] Sag, I.A., A semantic theory of NP-movement dependencies, in: P. Jacobson

and G.K. Pullum Edts. *The nature of syntactic representation*, D. Reidel 1982.

[28] Saint-Dizier, P., Expression of Features in Logic-Based Grammars, *Computational Intelligence*, 1986, vol. 2, no. 1.

[29] Saint-Dizier, P., An approach to Natural language semantics in logic programming, to appear, *The journal of Logic Programming*, 1986.

[30] Sebillot, P., Saint-Dizier, P., Un processes de filtrage par arbres syntaxiques dans les grammaires dicontinues, *Proceedings Journees PROLOG-CNET, 1986*

[31] Sharp, R., A model of Grammar Based on Principles of Government and Binding, Ms.Sc., Univ. of British Columbia, Canada, 1985.

[32] Schieber, S., Stucky, S.U., Uszkoreit, H., Robinson, J., Formal Constraints on Meta-Rules, *Proceedings of 21st ACL meeting*, 1983.

[33] Stabler, E., Restricting Logic-Grammars. Quintus Computer Systems report, 1986.

APPENDIX

Expression of Ross' constraints in Discontinuous Grammars

In the DG rule context, the expression of Ross' constraints is divided into four sub-problems :

(1) The way in which to memorize the derivational context of each symbol of a rule. The derivational context of symbols will be later compared to the contexts R_i of constraints.

(2) The definition of the domain, for a given constraint, in which any constituent of this domain may move without restriction. We call this domain the C-domain of the constraint.

(3) The detection that a constituent of a given C-domain moves out of this domain.

(4) The detection that a constraint is applicable and its expression in a rule.

1_ Constructing a symbol's derivational context

As we have seen, the pictorial representation of a sentence 's derivation in DGs is usually a graph, since some rules may have more than one left-hand side symbol. However, for the purpose of examining constraint applicability, we shall only keep track of a tree. We make the assumption that only one parent is relevant to a symbol's derivational context. In those cases where a rule has more than one left-hand side symbol, the grammar writer must specify a unique parent for each symbol that the rule introduces. This must be the parent whose grammatical function is the most related to the symbol's. If the symbol is simply rewritten, the system will assume its own left-hand-side occurrence as its parent. Skips are ignored, since no change is introduced into them by the rule. For instance, in the rule shown in section 2.3:

rel_marker(X), skip(G), trace(X,P1,P) –> rel_pronoun, skip(G).

the grammar writer must recognize the np trace as the constituent that lends its grammatical function to the relative pronoun. S/he indicates this by adding a parent indicator as a first argument of the rel-pronoun symbol:

rel_marker(X), skip(G), trace(X,P1,P) –> rel_pronoun(par(trace(X,P1,P))), skip(G).

This is an artificial simplification made in view of ease of implementation, and which seems appropriate for a great number of cases. However, we do

not exclude the possibility for other parent symbols to intervene in a symbol's inheritance of other important features. We only make the working hypothesis that grammatical function alone will be enough to establish the relevant links between constituents that will allow us to decide upon constraint applicability.

Given this hypothesis, we can extract the derivational tree of a sentence from its derivational graph: for each constituent with multiple parents in the derivation graph, we delete all links coming from the constituents not declared, either explicitly or implicitly, as parents.

Consider, for instance, the following grammar:

ax --> bx, cx, dx, ex.

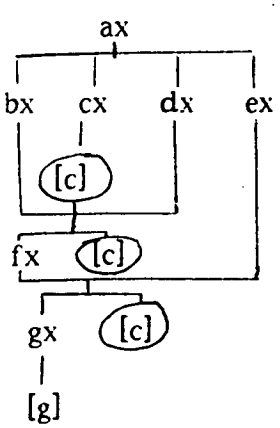
bx, skip(X), dx --> fx(par(bx)) skip(X).

fx, skip(X),ex --> gx(par(ex)) skip(X).

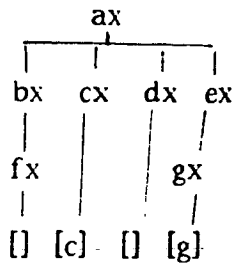
cx --> [c].
gx --> [g].
ax --> [a].

and the string: [g,c]

with derivational graph:



the function derivational tree is:



The derivational context of a symbol (noted dc(X)) is defined as its derivation tree minus the subtree of which this symbol is the root. We will use it to compare it with a constraint's context(previously denoted by Ri). We next show how to construct the derivational tree, which can then be used to examine derivational contexts.

Derivation trees can be constructed by adding one more argument to each symbol. This can be statically done by a preprocessor. This extra argument is a list, the first element of which represents the derivation tree up to the symbol itself, and the second element represents the partial structure that this rule contributes to the tree. For instance, the rules:

```
sentence(P) - > np(X,P1,P), vp(X,P1).

np(X,P1,P) --> det(X,P2,P1,P), noun(X,P3), relative(X,P3,P2).

rel-marker(X), skip(G), trace(X,P1,P) --> rel_pronoun(par(trace(X,P1,P))), skip(G).
```

would automatically be transformed into:

```
sentence(S,P) --> np([sent(NP,VP),NP],X,P1,P)
          vp([sent(NP,VP),VP],X,P1).

np([T,np(Det,N,Rel)],X,P1,P) -->
   det([T,Det],X,P2,P1,P)
   noun([T,N],X,P1)
   relative([T,Rel],X,P3,P2).

rel-marker([T,[]],X) skip(G) trace([T1,trace(Relp)],X,P1,P) -->
   rel-pronoun([T1,Relp]), skip(G).
```

It should be noticed that the derivational context is built step by step as soon as a new rule is activated.

2_ C-domain of a constraint

For each constraint Ci, we define a **C-domain** D_i, out of which no constituent can move. The C-domain is the subtree with root node N_i. This notion of C-domain is very similar to that of c-domain in LFG, but C-domains are not here fixed once for all in rules.

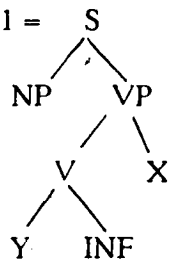
In the first example in section 5.1, the C-domain D_i in which A (element of D_i) may move without restriction is the domain dominated by the node S.

3_ Detection of a constituent's movement out of a given domain

We say that a symbol A_i (that represents a constituent) belongs to a domain D_i with root node N_i iff N_i is an ascendant of A_i in its derivational context $dc(A_i)$.

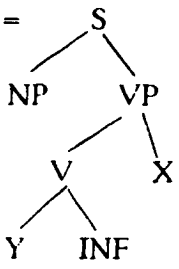
Thus, two constituents X and Y belong to the same domain Dj with root node Nj iff:
(1) Nj is in the derivational contexts of both X and Y, and
(2) The occurrence of Nj in dc(X) has the same derivational context as its occurrence in dc(Y). This latter constraint is necessary because several occurrences of the symbol N_i may appear in the parse tree of a sentence.
For instance, if we have X with derivational context:

dc1 =



and Y, with derivational context:

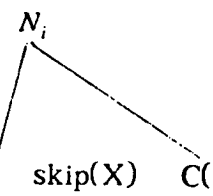
dc2 =



belong to the same domain Dk with root node VP

but not to the same domain if Dm has the root node V

Given these definitions, we can now characterize the movement of a constituent out of a given domain. Let $Ci(R_i, N_i)$ be a constraint and Gi a rule with movements. Then, at the level of rule Gi, the leaves of the domain D_i of Ci are the set of symbols in the left hand side of G_i that have N_i in their respective derivational contexts. For instance, we may have:



Gi: A B skip(X) C(1) skip(Y) D(1) –> ....

Iff B and C have N_i in their derivational contexts.
C and D are indexed by (1) because this rule describes a movement in which they are involved.

The constraint is then considered as being relevant for Gi only if the path in the tree from the root to N_i (i.e. the derivational context of N_i) unifies with the context R_i of the constraint.

However, the following particular cases will invalidate the relevance of the constraint:

(a) N_i dominates no element in Gi marked as moving,
(b) N_i dominates all the elements of the left hand part of Gi.

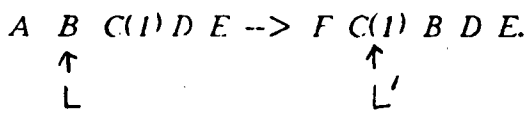
The next point is to locate D_i in the right hand part of the rule, using symbols in Gi as indexes:

- directly if they are not subject to reduction,
- via the parent notation if they are reduced: the parent notation is viewed as a chaining.

More precisely, the domain D_i is delimited in the left hand side of a rule by a left and a right border materialized by symbols we call L and R. In our previous example, L= B and R = C.

The problem is to define L' and R', respectively the left and the right borders of D'_i in the right hand side of Gi. We distinguish two cases, depending on whether L or R are marked as being moved elements or not. L and R behave in a symmetric way, thus, we will consider here L, the result being the same for R, substituting "right" for "left".

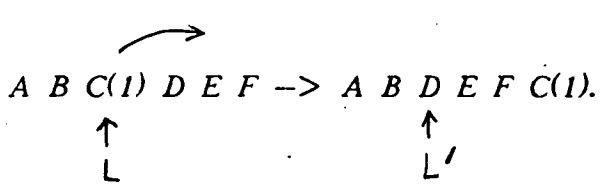
●(1) If L is not marked as a moving symbol, it remains the border in the right hand part of the rule. However, if an element of D_i is moved at the immediate left of L then this element becomes the border if it keeps its filiation (even if it is subject to reduction, expansion or duplication), or if it inherits of the derivational context of another element in D_i. For example, if we have the following case:



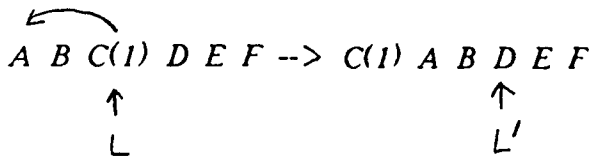
C(1) belongs to D_i and becomes L' since it keeps its filiation, i.e. it does not inherit of the derivational context of an element out of D_i via the "parent" notation. This is the case, for instance, for the topicalization of an adjective in an subject NP. Conversely, C(1) no longer belongs to D_i if it inherits of the filiation of an element out of D_i, as for instance, if it is noted: C(1,par(A)). This may be the case, for instance, in raising operations or wh-movements.

●(2) L is marked as being a moving constituent. If L has a movement either to the right or to the left, then L' is the element immediately on the right of L. For instance, for L' we have the following cases :

right movement :



left movement:

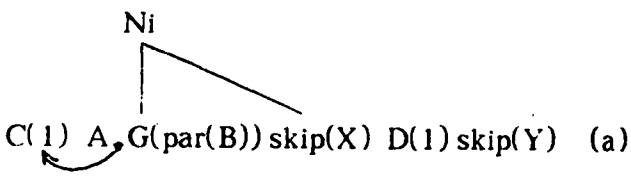


Condition (1) or (2) are applied recursively for each side of the domain until all elements marked as moving inside D_i in the left hand part of Gi have been considered.

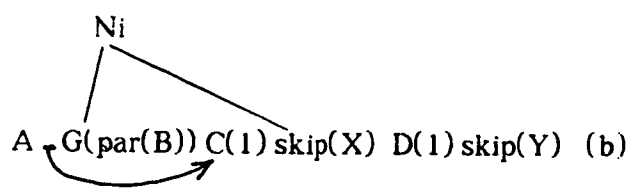
4_ Applicability of a DG rule:

The final stage is to detect whether a moving element in D_i is extraposed out of D_i. First, when several movements are described within a rule, and, thus, indexed by different integers, each movement is treated separately, in the increasing order of indexes.

If we consider a given movement, then a constituent moves out of D_i if it is outside D'_i, delimited by L' and R'. For instance, we may have as right hand parts for the rule G_i above:



or:



in (a), C(1) moves out of D_i whereas in (b) it moves in D_i.

Then, if an element moves out of the domain D_i the rule is blocked.

5_ Properties and implementation:

The PROLOG program that transforms rules with skips into a set

of Horn clauses is described in detail in [8]. To this program, we have added a second module which is executed before transforming DG rules into Horn clauses and which is completely independent of this transformation [30].

The goal of this module is, given a DG rule, to automatically add (1) an argument in which the derivational context is built and (2) a PROLOG-call "verif_const" with adequate arguments to control movements. Constraints themselves are given once and for all, and represented by PROLOG facts. They are stated independently of the grammar being described, since they are independent of the sublanguage being dealt with.

Given a rule with moving elements, the first task of this program is to create a new argument for each symbol in the rule and to specify in this argument how the rule contributes to the construction of the derivational context of the corresponding symbol. This point has already been explained in section 1. Note that to create the derivational context, we have to dynamically generate new variables.

The second task, which is, in fact, performed at the same time as the first one, is the adjunction of the PROLOG-call verif_const at the end of the left hand part of the rule Gi. [32]. This predicate has two arguments which respectively contain the left and right hand parts of the rule. However, in verif_const symbols only contain two arguments (the others being discarded because they are either useless for controlling movements or redundant (e.g. the parent argument)). These arguments represent:

- the movement argument (= nil if the symbol is not involved in any movement).
- the derivational context.

Skips are recopied unchanged.
For instance, consider the example given at the end of paragraph 1 of this appendix, where we have the following rule:

```
rel_marker(X), skip(G), trace(1,X1,P1,P) →
        rel_pronoun(1,par(trace(X,P1,P))), skip(G).
```

This rule becomes:

```
rel_marker([T|[]],X), skip(G), trace([T1|trace(Relp)],X,P1,P)
 {verif_const(rel_marker(nil,[T|[]]).skip(G).trace(1,[T1|trace(Relp)]),
          rel_pronoun(1,[T1|Relp]).skip(G))}   →
rel_pronoun([T1|Relp]), skip(G).
```

6_ Performances

For each constraint, the PROLOG call verif_const performs the process described in sections 3 and 4. In a first implementation [30], verif_const is executed sequentially for each constraint. It fails if a

constraint is satisfied. The following performances were obtained using the grammar given in sections 2.2 and 2.3, to which we have added a third skipping rule to allow adjective movements. We consider two sentences involving rules with skips:
A: "The man that Jill saw is here."
B: "The man is here that Jill saw."
and a database of constraints ranging from 0 to 5 constraints. Times are given in milliseconds. the systems used is a 4 processor HB-68 with a Multics system.

number of constraints	A	B
0	833	857
1	815	823
2	780	791
3	767	771
4	819	788
5	791	794

Results show that the adjunction of the tree filtering constraints save from 10 to 20% of the processing time. In fact, constraints block derivations as soon as an incorrect configuration is detected in the parse tree being built. The expression of constraints in rules does not entail the adjunction of many time consuming devices. As, most of the time, a syntactic tree is built, the only time consuming device is the predicate verif_const.

The best performances are obtained with 3 constraints. The efficiency with only one constraint is not so good because few ill-formed tree configurations are filtered out. The parser also becomes less efficient when more than 4 constraints are used. This is due to irrelevant constraints that cannot be discarded a priori.

It is possible to overcome this problem of variations of efficiency if constraints are independent from each other. Results are expected from a simulated parallel execution we are now carrying out. However, it should be noted that the high level of the constraint approach usually results in very few constraints, particularly if the framework used is Government and Binding theory.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique