Some Aspects of GB-Parsing within the CLP Framework

Nabil Hathout
IRIT – Université Paul Sabatier
118, route de Narbonne. F-31062 Toulouse Cédex.
e-mail: hathout@irit.fr

Using a non trivial linguistic theory is essential in computational linguistics if one aims at building a non trivial natural language processing system, the answers of which are reliable for a large variety of sentences. In consequence, it is necessary to use advanced computational tools for this system to be well designed and efficient. These reflexions has led us to base our work on the well-known Government and Binding theory (hereafter GB theory) proposed by Chomsky (1981, 1982, 1986a, 1986b) and to adopt a constraint logic programming approach (Cohen 1990, Colmerauer 1990, Jaffar and Lassez 1987a). The main caracteristics of GB theory is that it is modular and parametrizable, that is, composed of a small set of principles which have a small number of parameters. According to this theory, a sentence is grammatical if and only if its various representations do not violate any of the theory principles. This conception of the grammaticality is very close to that of well-formedness in constraint logic programming (hereafter CLP). A CLP program is a set of constrained Horn clauses and a CLP goal is a constrained formula which succeeds if and only if all the constraints of the goal and of the clauses used to prove it, are satisfied. In other words, the goal succeeds if it is the logical consequence of the program clauses and if it does not violate any of their constraints.

In the following, we first present some basic components of GB theory, then we address the point of how GB theory may be used to design parsers or generators. We end the first section by describing the architecture of a GB-parser. In the second section, we show how some GB subtheories can be modeled as constraints upon syntactic processes and how to integrate them into a CLP framework.

1 Parsing with GB Theory

We consider, for expository purpose, that the GB modules fall into two groups: productive and restrictive. This may appear a bit artificial, but it is costless, since it does not change in any manner neither the (GB) principles nor the theory. We regard as productive, the modules used to generate representations, namely the Projection Principle, the Functional Selection and the various assignment mechanisms (indexation, Case marking, θ-marking...). These modules are constrained by the restrictive ones (i.e. the others) in different ways. For instance, X$'$-theory is a constraint upon the syntactic categories (i.e. the outputs of the productive modules), which imposes the form they must have. The core idea of our story, is

Presented at the ICLP Workshop on Advanced Logic Programming Tools and Formalisms for Language Processing

to have an architecture in which the productive modules are implemented as Horn clauses, and the restrictive ones as constraints associated with these clauses.

In this section, we give an overview of three basic GB modules: X′-theory, the Projection Principle and its extension, the Functional Selection. Afterwards, we discuss the advantages of processing from right to left. We end by showing how the previous three modules may induce the structure of a GB-parser.

1.1 X′-Theory

In GB theory, sentences are represented by trees which describe syntactic categories. In the latest versions of the theory, one can discern two kinds of (atomic) categories: the major ones: N, V, A and P, and the functional ones (often called "non-lexical categories"): C (complementizer), I (inflection), D (determiner), Neg (negation), Agr (agreement), T (tense)... All these categories, called heads, project into phrases which are subject to X′-theory. In the following, we only consider, among the functional categories, C, I and D, because we do not, for the time being, comprehensively deal with head movements.

The version of X′-theory usually adopted by linguists is the one proposed by Stowell (1981:70). It is composed of the five following principles:

(1) (*i*) Every phrase is endocentric.

(*ii*) Specifiers appear at XP level.
Subcategorized complements appear within X′.

(*iii*) The head always appears adjacent to one boundary of X′.

(*iv*) The head term is one bar level lower than the immediately dominating phrasal node.

(*v*) Only maximal projections occur as non-head terms within a phrase.

These principles can be translated straightforwardly as computational constraints on the representations. Actually, they may be more efficiently used if we reformulate them in such a way that we could check them locally. We then obtain:

(2) (*i*) A category is either a head or a projection.

(*ii*) If a category is a specifier, then its immediately dominating node is an XP.
If a category is a subcategorized complement, then its immediately dominating node is an X′.

(*iii*) If a category is a head, then it is either the first or the last daughter of an X′-projection.

(*iv*) If a category is a head, then it projects into an X′-projection.

(*v*) A category either projects into another one or is an XP[1].

[1]This can be reformulated as follow:
principle_1(X): head(X) ∨ projection(X,_).
principle_2a(X): specifier(X,Y) ⟹ daughter(X,Y) ∧ xp(Y).
principle_2b(X): complement(X,Y) ⟹ daughter(X,Y) ∧ x-bar(Y).
principle_3(X): head(X) ⟹ first_daugter(X,_) ∨ last_daugter(X,_).
principle_4(X): head(X) ⟹ projection(X,Y) ∧ x-bar(Y).
principle_5(X): projection(X,_) ∨ xp(X).

But this description has a shortcoming: it is very insufficient. If we wish to utterly describe the X′-trees, we must at least complete (2) so as (*i*) to ensure that the representations are trees, that each head projects into a phrase... and (*ii*) to describe relations like relation... (Jackendoff 1977). We would then have a very interesting model of X′-theory, very close to the linguistic description, which permits of a straightforward running of the X′-principles in the parser. Nevertheless, we will not adopt this approach because it would be rather difficult to implement, and above all, very inefficient. We rather choose a more traditional alternative, that is, to compile the X′-principles into an X′-grammar schema (Saint-Dizier 1988). This schema has been a bit modified, in order to take into account the convention, proposed by Chomsky (1986a), that the X′-projections which are single daughters must be deleted. It is thus composed by the following rules:

(3) $X' \rightarrow X^0$, *complement*$^+$
$XP \rightarrow X^0$
$XP \rightarrow X^0$, *complement*$^+$
$XP \rightarrow$ *specifier*, X^0
$XP \rightarrow$ *specifier*, X'

where X stands for any syntactic category. In this schema, X^0 is the only terminal symbol, *complement*$^+$ stands for one or many maximal projections, i.e. XPs, and *specifier* for a maximal projection. We can see that (3) does not cope with adjuncts since adjunction does not belong to X′-theory (it rather belongs to the Movement Theory). Yet, it can be described by a similar schema:

(4) $XP \rightarrow$ *adjunct*, XP
$XP \rightarrow$ XP, *adjunct*

where *adjunct* stands for a maximal projection. We will resume this discussion later (see 2.3, page 10). Let us notice by the way, that in these rules, XP on the left is exactly the same as the one on the right, except that it has an extra daughter: the adjunct.

1.2 Phrase Projection

As we just have seen, X′-theory is a set of constraints on the phrase structures. These phrases may be regarded as the result of the Projection Principle and its extension: the Functional Selection. The former deals with major categories, and the latter with functional ones.

1.2.1 The Major Categories

The structures of major category phrases depend on their lexical entries. These entries describe the θ-roles they assign, and the categories they subcategorize for. For instance, the verb *eat* assigns an *agent* external θ-role (underlined) and a *theme* internal θ-role, and subcategorizes for a *DP* direct object (in the subcategorization list, '_' stands for the verb position):

(5) eat: <agent, theme> [_ DP]

Major categories are projected by means of the Projection Principle. More precisely, this principle projects thematic relations from the lexicon into the syntax, namely into each representation level. It maps each major category into a phrase representation (i.e. the part of the syntactic tree which corresponds to the phrase headed by this category), which contains an argument position for each θ-role of the lexical entry (the internal θ-roles correspond to complements, and the external θ-role corresponds to a specifier). Besides, the only phrases that may occur in the complement positions are those of the subcategorization list. For example, the phrase representation that corresponds to the lexical entry (5) is:

(6) $[_{VP}$ Spec $[_{V'}$ $[_{V}$ eat] $[_{DP}$ ...]]]

1.2.2 The Functional Categories

The functional categories are also projected into phrase representations. However, the relation between a functional category and its complement is completely different from that between a major category and its θ-marked complement. Actually, there are syntactic relations between all heads and their complements and adjuncts, by which those complements and adjuncts are licenced. These relations divide into two classes (Abney 1987:55):

- thematic relations, on the one hand, include θ-marking, and the relation by which adjuncts are licenced.
- Functional Selection, on the other hand, is the relation between a functional head and its complement.

Abney (1987:64–65) proposes some properties which characterize functional heads:

1. Functional heads permits only one complement which is not an argument (i.e. neither a noun phrase, nor a prepositional phrase, nor a subordinate clause).
2. Functional heads cannot be separated from their complement. Functional Selection can be regarded as a near bijective relation between them and their complements: an NP is always a D complement, a VP is always an I complement and an IP is always a C complement. This relation can thus be used in both directions to build representations.
3. Functional elements lack descriptive content. Semantically, they do not describe a distinct object from that described by their complement: they contribute to the interpretation of their complement rather than pick out a class of object.

As a result of the near bijective nature of the Functional Selection, the structures of the functional phrases do hardly depend on their context (i.e. of the sentence in which they occur), because of the great regularity of their specifier distribution extensively studied by Abney (1987). As far as C, I and D are concerned, we can summarize these considerations as:

- CP may either be $[_{CP}$ Spec $[_{C'}$ C IP]], or $[_{CP}$ C IP] according to the occurrence of a wh-phrase in the clause.
- IP is always $[_{IP}$ Spec $[_{I'}$ I VP]].
- DP may either be $[_{DP}$ D NP], or $[_{DP}$ D] when it is a pronoun (at least in French).

Functional phrases can thus be built in accordance with these five patterns which may be regarded as grammatical rules describing these phrases. At last, let us notice that in a bottom-up approach, it is the complement which triggers the building of the functional phrase by which it is selected.

1.3 Processing from Right to Left

As regards the processing stategy, we believe that the syntactic processing of SVO languages such as French should be bottom-up and from right to left. Even though we agree that this approach is not very natural since humans process language from left to right, we claim that, inasmuch as GB theory is used, processing from right to left is more adequate than processing in the opposite direction. If one assumes that parsing and generation are very similar (as we do), in that extent that both aim at building a syntactic representation starting either from a sentence (a string of words) or from a logic formula, then our proposal consists in treating these inputs from right to left. The main justification of our proposal is that the syntactic representations have usually the following shape:

(7) [. . . [. . . [. . . [. . . [. . . [. . .]]]]]

Thus, right to left processing comes down to building the representations from the inner to the outer. Moreover, many categories do not have specifiers, and thus do not contain any constituent on the left of their heads. (The only categories which may have specifiers are VP, IP and CP, and the distribution of their specifiers is rather well known.) So, with right to left parsing, we compute the representations of the complements before that of the phrase in which they occur. Also, the head is the *last* (i.e. the left most) element of the phrase to be treated. This results in several advantages:

1. The partial representations we build during the processes are often closed, that is without unknown parts or holes. We can, therefore, locally check the satisfiability of the constraints of these representations and locally solve them.

2. We can more efficiently choose the right lexical entry corresponding to the current word without resorting to any look-ahead mechanism, since we exactly know all the available complements of that head. As a result, we also have a good knowledge of the external arguments of these heads; we may thus expect them, which makes the process better driven.

3. The empty categories positions can be determined rather easily: each time the Projection Principle or the Functional Selection predicts a category which no word in the sentence may correspond to. Since phrases usually move to the left (except heavy DP-shifts and PP-extrapositions from NPs), we may expect the empty categories antecedents also, i.e. their binder (DP-traces) or their binding operator (wh-traces).

4. But the main advantage of right to left processing is that it permits coping with trace and movement theories very efficiently. In effect, we know all the traces that occur in the partial representations, and we can determine rather early the barriers for these traces. As a consequence, we can easily (and early) check the Empty Category Principle (hereafter ECP) and the Subjacency Condition for each of them. We can then determine if intermediate traces have to be adjoined, or if it is not necessary, or if the sentence is agrammatical (i.e. if it violates one of these two conditions).

1.4 A GB-Parser Structure

Before we outline the parser structure, we will first discuss the nature of the parser output. This leads us to give an overview of the various representation levels of the theory.

1.4.1 Representation Levels

In GB theory, each sentence is represented at four representation levels (see (Haegeman 1991, Lasnik and Uriagereka 1988, Sells 1985) for more details):

- The deep form of a sentence is called D-Structure. It encodes the predicate-argument and the thematic relations the sentence. In other words, it reflects the lexical properties of the sentence constituents.
- The surface form, called S-Structure, reflects more superficial properties of the sentence and accounts for the surface ordering of the sentence constituents. S-Structure is related to D-Structure via the move-α relation.
- The logical form, called LF, represents the logico-semantic properties of the sentence. It is an intermediate step between S-Structure and the semantic representation advocated by semanticists. LF is derived from S-Structure by means of the affect-α relation. (Where affect-α = move-α + delete-α)
- The phonetic form, called PF, encodes the surface properties of the sentence, and is derived from S-Structure.

Only the three first representation levels are syntactic since they are subject to syntactic principles such as the Projection Principle, the θ-criterion or ECP. On the other hand, PF is not regarded as syntactic and is of very little interest for us (in this paper).

The result produced by the GB-parser we describe in the next subsection are S-Structure representations. Actually, S-Structure is the most interesting level because, among the three syntactic levels, it is the closest to the surface representation of the sentence. Moreover, the bijective nature of move-α entails that we can recover the D-Structure from the S-Structure[2]; we only have to restore each constituent in its most embedded trace position. Also, D-Structure representations are not of great interest, since every principle that applies at D-Structure apply also at S-Structure.

Since only one representation is produced, we indeed regard movement (i.e. move-α relation) as a mere co-indexing relation between an empty category and an antecedent. This relation has (still) to obey the subjacency condition. At last, we can check ECP without explicitely computing an LF representation of the sentence: we only simulate affect-α at S-Structure level.

1.4.2 The parse architecture

The parser structure we present is rather simple and implements more or less the modelling idea described in the introduction of this section (page 1). Its productive part is composed of a projection module and an integration module, whereas its restrictive part consists of constraint solvers.

[2]Parasitic gaps do not pose any problem if we adopt Chomsky's analysis (1986a) that are bound by independant empty operators.

The projection module builds partial representations which are the projections of the sentence words. They represent the categories produced by the projection principle possibly extended by those induced by Functional Selection. For instance, the partial representation of a noun phrase is a DP, generated via Functional Selection, which complement is an NP generated via the projection principle. This module thus implements the projection principle, the Functional Selection and some assignment devices. From a computational point of view, it is worth noting that a lazy inheritance strategy is used to build these representations. For example, information relative to the case, the θ-role and the binding properties of a noun appears only in the top most DP category of its representation. This considerably reduces the size of the representations and makes them more readable and informative since information only appears where it is relevant.

These representations are then given to the integration module which assembles them. It tries to fill those of their positions whose content is undefined. According to the thematic relations (see 1.2.2), it sets in these positions either a partial representation previously built or an empty category (if it is allowed). Besides, this module manages a stack of the not yet integrated representations which ensures that the word ordering is respected.

Along with the integration process, several constraints are posited upon the partial representations. They enforce GB principles such as ECP or the binding principles, and are independently solved by means of specific constraint solvers. During the parsing process, all these constraints interact with each other, which makes this parser rather accurate since its behavior is very similar to the parsing process described by the linguistic theory.

2 GB Principles as CLP Constraints

The use of constraints constitutes an interesting development of logic programming (hereafter LP). We present, in this section, their main contributions, and the way we use them in the previously described NL system. Also, we propose CLP modelling of some GB subtheories such as Binding Theory.

2.1 The CLP Approach

Constraints contributions to logic programming are many. First of all, the use of constraints improves LP expressiveness because of the introduction of new computation domains (such as boolean domain, arithmetic domain, finite domains...) beside the usual Herbrand's one (Jaffar and Lassez 1987a, Jaffar and Lassez 1987b). The user can thus handle directely the objects of the intended domain (i.e. the domain of discourse), as opposed to having them encoded as Herbrand's terms. Similarily, the constraints describe properties directely in the intended domain, which results in a saving of naturalness. Also, we can, by means of constraints, represent properties implicitely and define objects intensionaly. We then have available not only the objects, but also the properties which characterize them.

Moreover, constraints enhance LP declarativeness because of the resolution method. In CLP systems, constraints are solved by independant solvers which are integrated at unification level. Constraints can thus be set at any time, contrarily to Prolog in which constraints (such as arithmetic predicates) can only be invoked if certain parameters are completely known (Colmerauer 1990). So, constraints make LP become (more) equational and make the variable behave as genuine mathematical unknowns.

But the main contribution of the CLP languages is the active use of the constraints, which results in a better efficiency of the LP systems (Dincbas *et al.* 1988, Van Hentenrick 1989). Whereas Prolog uses the constraints according to the "generate and test" schema, CLP systems use them at each (Robinson's) resolution step, to reduce the search space *a priori* that is, before the generation of all their values. This is achieved by reducing the variable value domains via simplification of the constraint system. Also, with an active use of constraints, backtracking happends earlier (as soon as the constraint system becomes inconsistent).

The parser we presented in section 1.4.2, also uses actively constraints. As we said, some GB principles can be implemented as constraints associated with the integration process. These constraints are not handled directly at unification level for efficiency reasons. (Actually, CLP meta-interpreters, built on the top of a Prolog system, are not very efficient.) However, we simulate the active constraint handling in our NL system, as it is done in CLP meta-interpreters (Cohen 1990). In the integration process, new constraints are posited at each filling step. Moreover, the only operation which modifies the partial representations is that of filling. So, we content ourselves with checking the satisfiability of the constraint system and simplifying it (i.e. solving it) at each filling step. This is illustrated by the following definition of the predicate *fill*. Its arguments are respectively the representation that must be integrated, the list of the positions with undefined contents, which may receive that representation, and the input and output constraint systems.

```
fill(Rep,[],Ctr,Ctr).
fill(Rep,[Pos|PosList],CtrIn,CtrOut):-
constraints_on_filling(Rep,Pos,NewCtr),
solve(NewCtr,CtrIn,CtrOut),
set(Rep,Pos).
fill(Rep,[Pos|PosList],CtrIn,CtrOut):-
constraints_on_filling(trace,Pos,NewCtr),
solve(NewCtr,CtrIn,CtrInterm),
set(trace,Pos),
fill(Rep,PosList,CtrInterm,CtrOut).
```

The heart of this predicate is the procedure *solve* which merges two sets of constraints: the constraints *NewCtr* introduced by the new filling and the previous constraints *CtrIn*. If the resulting set of constraints is unsatisfiable (i.e. if there is no solution to this constraint system), the procedure fails; otherwise, it simplify the resulting constraints and binds any variables which have been constrained to a unique value. *constraints_on_filling* provides, for each couple (*Rep*,*Pos*), the set of constraints corresponding to the filling of *Pos* with *Rep*. As for *set*, we only introduced it for expository purpose, since it is actually subsumed by *solve*. It is worth noting that *fill* can be used to complete the current representation (i.e. filling on the right) as well as to complete a representation of the stack (i.e. filling on the left): we have only to order *PosList* in the right way.

2.2 Binding Theory

In this section, we propose a modelling of Binding Theory as a set of constraints on indexation, or more precisely on co-indexation. Actually, every category has an index, i.e. an integer, which characterizes the referent associated with this category bi-univocally. So, two categories which correspond to a same object or event, share the same index.

2.2.1 The Binding Conditions

Binding Theory deals with the noun phrase interpretation. It is composed of three conditions which state, according to a [±a, ±p] typology ('a' stands for anaphoric and 'p' for pronominal), that noun phrases must or must not have antecedents within certain local domains. This typology divides noun phrases (i.e. DPs) into four classes:

1. The only category with [+a, +p] features is the empty category PRO.
2. The anaphors have [+a, −p] features and consist of reflexives, such as *himself*, of reciprocals, such as *each other*, and of DP-traces.
3. Pronouns such as *he* and *him* are pronominals and have the features [−a, +p].
4. Names (i.e. fully referencial nouns) such as *the cat* and *Mary*, and wh-traces are R-expressions and have [−a, −p] features.

The other basic notion of Binding Theory is the binding relation defined as: X binds Y iff X c-commands Y and X is co-indexed with Y (for more details, see (Haegeman 1991, Lasnik and Uriagereka 1988)). The three conditions of binding are:

- Condition A: an anaphora must be bound in its governing category. This can be reformulated as: there must be at least one category which occurs in an anaphora governing category, which c-commands that anaphora, and is co-indexed with it. In other words, if i is the index of an anaphora X, and if $j_1, \ldots, j_n$ are the indexes of the categories c-commanding X and dominated by the governing category of X, then i must belong to the set $\{j_1, \ldots, j_n\}$. So, with respect to X, Condition A can be expressed by means of a constraint $\mathit{belongs}(i, [j_1, \ldots, j_n])$. We go back over the computational properties of this constraint below.
- Condition B: a pronominal must be free in its governing category. This can be detailed as: there must not be any category occuring in the governing category of a pronominal, which c-commands that pronominal and is co-indexed with it. In other words, if i is the index of a pronominal X, and if $j_1, \ldots, j_n$ are the indexes of the categories c-commanding X and dominated by the governing category of X, then i must not belong to $\{j_1, \ldots, j_n\}$, that is, i must be different from all the members of $\{j_1, \ldots, j_n\}$. So, we can express Condition B for the pronominal X, via the constraint $\mathit{different}(i, [j_1, \ldots, j_n])$.
- Condition C: an R-expression must be A-free. This condition is similar to Condition B, and states there must not be (in the whole sentence) any category occuring in an A-position, which c-commands an R-expression and is co-indexed with it. So, if i is the index of an R-expression X, and if $j_1, \ldots, j_n$ are the indexes of the categories c-commanding X and occuring in A-positions, then i must not belong to $\{j_1, \ldots, j_n\}$, that is, i must be different from all the members of $\{j_1, \ldots, j_n\}$; this can be expressed via the constraint $\mathit{different}(i, [j_1, \ldots, j_n])$.

2.2.2 Constraint Propagation Techniques

In this second part, we discuss the computational aspects of the previous constraints, and more generaly of the entire indexation process. Since indexation is a constraint satisfaction problem (hereafter CSP), we present, at first, the CSPs along with their resolution methods.

A constraint satisfaction problem can be defined as a couple (X,C) where X is a set of variables, and where C is a set of constraints on variables of subsets of X. A constraint on a set of variables is a restriction on the values they can take simultaneously. The task in a CSP is to find a solution, namely an assignment of one value to each variable of X, such that all the constraints of C are satisfied. Usually, X is a finite set of variables $\{x_1, \ldots, x_n\}$ which take their values from finite domains $d_1, \ldots, d_n$. A constraint $c(x_{\alpha_1}, \ldots, x_{\alpha_k})$ on k variables of X is then a subset of the Cartesian product $d_{\alpha_1} \times \ldots \times d_{\alpha_k}$ which specifies which values of the variables are compatible with each other. So, a solution of the CSP is any member of the intersection of the constraints of C.

There are mainly two ways for solving CSPs: backtracking and constraint propagation techniques (Van Hentenrick 1989). The resolution based on backtracking consists in assigning values to variables of X and testing if this assignment is a solution or if it can be extended to a solution. The backtracking approach is thus nothing more than solving CSPs by using a Prolog search procedure, the search space being the Cartesian product $d_1 \times \ldots \times d_n$. On the other hand, constraint propagation techniques (also called consistency techniques) are based on the idea of *a priori* pruning, that is, using the constraints to reduce the search space before discovering a failure. The pruning is achieved by spending more time at each node of the search tree removing every combination of values that cannot appear in a solution (Freuder 1978, Mackworth 1987).

Indexation consists of assigning an index to each category of a representation. As the number of indexes that occur in a representation is at most equal to the number of heads (which is finite and known), all the constraints involved in this process are constraints on finite domain variables. So, indexation is a typical CSP which may be solved by means of constraint propagation techniques. In this problem, the variables represent the indexes of the representation categories. So, we can take as domain of these values, the set $\{1, \ldots, n\}$ where n is the number of heads that occurs in the representation.

The constraint $\mathit{belongs}(i, [j_1, \ldots, j_n])$ means that i must be equal to (i.e. unifiable with) at least one of the members of the list $[j_1, \ldots, j_n]$. Let us note that *belongs* is not compositional, that is $\mathit{belongs}(i, L_1) \wedge \mathit{belongs}(i, L_2) \not\Longleftrightarrow \mathit{belongs}(i, L_1 \cup L_2)$. This entails that a constraint *belongs* corresponding to a Condition A may only be set when all the indexes of the concerned categories are known. The constraint $\mathit{different}(i, [j_1, \ldots, j_n])$ states that i must be different of all the members of $[j_1, \ldots, j_n]$. As opposed to *belongs*, *different* is compositional and may be set in several steps.

Many other GB principles intervene into the indexation process, for instance Spec-head agreement... But the most important of them are those related to movement, such as the move-α relation, the Subjacency Condition and ECP. We can notice incidently that in our analysis, movement is regarded as a co-indexation relation between a trace and an antecedent. So, move-α is nothing but a constraint which imposes that a trace must be co-indexed with an antecedent (since each trace "comes" from somewhere). This may be expressed by means of the constraint *belongs*, however we will not do it because this co-indexation requirement is subsumed by Condition A for A-movements (DP-traces are anaphors) and by the operator-variable binding for A′-movements (wh-traces are variables).

2.3 Adjunction

In this section, we propose a CLP handling of adjunction. It is based on a representation mode of the maximal projections and allows all kinds of adjunction. In particular, we use it

to cope with ECP and with the Subjacency Condition since they may involve the adjunction of intermediate traces.

2.3.1 Open Categories

The adjunction schema (4) of page 3 is based on Chomsky's proposal that adjunction is only allowed to maximal projections which are not arguments (Chomsky 1986a:6). Furthermore, we consider the modifiers as adjuncts, which entails several problems, such as left modifiers adjunction. Since we parse from right to left, we do not know, when the phrase representation of a head is projected, whether there are modifiers on the left of that head. (Let us note that parsing in the opposite direction raises the dual problem of right modifiers adjunction.) But the main problem related to adjunction is that of intermediate traces which have to be adjoined to certain maximal projections so as to fullfil the Subjacency Condition or ECP. For instance, in the following sentence:

(8) $[_{CP}$ What$_i$ did $[_{IP}$ you $[_{VP}$ eat $t_i]]]$

VP and IP are barriers. This entails that the movement of the wh-element straight from the trace position to [Spec, CP] is excluded by Subjacency Condition since it forbids a movement to cross over more than one barrier. This movement must therefore be performed in two steps: the wh-phrase must first adjoin to VP and then move to [Spec, CP]. The resulting representation is then:

(9) $[_{CP}$ What$_i$ did $[_{IP}$ you $[_{VP}$ t'_i $[_{VP}$ eat $t_i]]]]$

In order to cope with adjunction, we propose to represent the maximal projections as *open* categories (which are indicated in the examples by the character $\rightleftharpoons$). More precisely, we only specify the top and the bottom of the maximal projections and we leave the segments which compose them partially undefined. The determination of these segments (i.e. of the adjunctions to these categories) results from the resolution of adjunction constraints. Initially, a maximal projection XP has the form (10a). If the resolution of some constraints implies that a category YP has to adjoin to XP, then XP becomes (10b). We can note that the segment is adjoined to the bottom, and that adjunction creates a new bottom category in order to permit XP not to change, with respect to the outer. When no more adjuncts may be added, we just have to make equal the top and the bottom of XP as in (10c), which closes the maximal projection.

(10) a. $[_{XP_1} \rightleftharpoons [_{XP_2} \ldots]]$

b. $[_{XP_1} \rightleftharpoons [_{XP_3}$ YP $[_{XP_2} \ldots]]]$

c. $[_{XP_1=XP_3}$ YP $[_{XP_2} \ldots]]$

For instance, the wh-movement of example (8) produces:

(11) a. $[_{CP_1} \rightleftharpoons [_{CP_2}$ What$_i$ did $[_{IP_1} \rightleftharpoons [_{IP_2}$ you $[_{VP_1} \rightleftharpoons [_{VP_2}$ eat $t_i]]]]]]$

b. $[_{CP_1} \rightleftharpoons [_{CP_2}$ What$_i$ did $[_{IP_1} \rightleftharpoons [_{IP_2}$ you $[_{VP_1} \rightleftharpoons [_{VP_3}$ t'_i $[_{VP_2}$ eat $t_i]]]]]]]$

c. $[_{CP_1=CP_2}$ What$_i$ did $[_{IP_1=IP_2}$ you $[_{VP_1=VP_3}$ t'_i $[_{VP_2}$ eat $t_i]]]]$

The introduction of open categories permits an easy handling of the overt adjuncts. Let us remember that there is an implicit word order constraint on the representations which imposes that the string of leaves of a representation must be identical to the string of the sentence words. In section 1.4.2, we proposed to implement this constraint by means of a stack of partial representations, managed by the integration process. We can therefore leave to this process the task of integrating the overt adjuncts; it only has to add new segments to the affected maximal projections.

2.3.2 ECP and Subjacency Condition

In the barrierhood framework, ECP and the Subjacency Condition are very similar in the extent that both involve the same kind of constraints on the representations.

The Subjacency Condition states that a movement must not cross over more than one barrier (Chomsky 1986a:28–31). In our analysis of movement, this comes down to imposing that a trace cannot be co-indexed with its antecedent if they are separated by more than one barrier. Since every trace must be bound by an antecedent, the Subjacency Condition can indeed be reformulated as: a trace must be bound by (and therefore co-indexed with) an antecedent from which it is not separated by more than one barrier.

As for the empty category principle, it states that traces must be properly governed. According to Chomsky (1986a:17), A properly governs B iff A θ-governs B (that is A governs and θ-marks B) or A antecedent-governs B (that is A governs and is co-indexed with B). Since θ-government does not involve any particular processing, we rather focus on antecedent-government. ECP may be interpreted as: a trace which is not θ-governed, must be co-indexed with a constituent (i.e. an antecedent) which governs it (i.e. which m-commands it, and is not separated from it by any barrier; for expository purpose, we assume that the Minimality Condition is respected). So, antecedent-government comes down to imposing, for each trace, the occurrence of a close enough co-indexed m-commanding constituent.

This leads us to express the Subjacency Condition and the antecedent-government by means of the following constraints, which will be associated to every trace t that will occur in the representation building:

- *subjacency*(t): there must be an antecedent (i.e. co-indexed c-commanding constituent) within the topmost category of the second barrier for t.
- *antecedent_government*(t): there must be an antecedent (i.e. co-indexed m-commanding constituent) within the topmost category of the first barrier for t.

The Subjacency constraint as well as that of antecedent-government can be satisfied in the same way:

1. either by co-indexing the trace with an overt constituent that already exists, and is close enough.
2. or by adjoining an intermediate trace (co-indexed with the previous one) to some maximal projection; this intermediate trace becomes the wanted antecedent.

The resolution of these constraints involves both co-indexation constraints and adjunction constraints resolution. However we must notice that *subjacency* and *antecedent_government* are not mere disjonctions of co-indexation and adjunction constraints, and that they need specific constraint solvers.

3 Conclusion

We have aimed througout this paper, at showing the suitability of the constraint logic programming approach to model GB theory and implement GB-based NL systems. We presented first an outlined structure of such a system and argued that we must, for efficiency reasons, use a right to left strategy. Then we presented a CLP modelling of Binding Theory and of some principles related to movement. At last, we proposed a representation mode which permits us to handle all kinds of adjunctions.

The work we have been presenting is still in progress and must be completed by the modelling of some other GB subtheories, such as Case Theory, θ-theory, Control Theory... However, the next stage will be to propose a CLP modelling of A- and A$'$-chain formation, and to design a complete set of constraints which permits us to cope with them. At last, let us emphasize again the adequacy of the CLP modelling because it allows the implementation of the GB subtheories as independant constraint solvers. Therefore, they can interact with each other, as intended by the linguistic theory.

Acknowledgement

Grateful thanks to Patrick Saint-Dizier, Pascal Amsili, Angeliek van Hout and Jacqueline Lecarme for their help and useful comments on earlier drafts.

References

Steven Peter Abney (1987). *The English Noun Phrase in its Sentential Aspect*. PhD thesis, MIT, Cambridge, Mass.

Noam Chomsky (1981). *Lectures on Government and Binding*. Foris, Dordrecht.

Noam Chomsky (1982). *Some Concepts and Consequences of the Theory of Government and Binding*. MIT Press, Cambridge, Mass. traduction Franaise : *La nouvelle syntaxe*, Seuil, Paris, 1987.

Noam Chomsky (1986a). *Barriers*. MIT Press, Cambridge, Mass.

Noam Chomsky (1986b). *Knowledge of Language: Its Nature, Origin and Use*. Preager, New-York.

Jacques Cohen (1990). Constraint logic programming languages. *Communications of the ACM*, 33(7):52–68.

Alain Colmerauer (1990). An introduction to prolog iii. *Communications of the ACM*, 33(7):69–90.

Mehmet Dincbas, Pascal Van Hentenrick, H. Simonis, A. Aggoun, Thomas Graf, and F. Berthier (1988). The constraint logic programming language chip. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 693–702, Tokyo.

E. C. Freuder (1978). Synthetizing constraint expressions. *Communications of the ACM*, 21(11):958–966.

Liliane Haegeman (1991). *Introduction to Government and Binding Theory*. Basil Blackwell, Oxford, UK.

Ray Jackendoff (1977). *$\bar{X}$-Syntax: A Study of Phrase Structure*. MIT Press, Cambridge, Mass.

Joxan Jaffar and Jean-Louis Lassez (1987a). Constraint logic programming. In *Proceedings of the Fourteenth ACM Symposium of the Principles of Programming Languages*, pages 111–119, Munich.

Joxan Jaffar and Jean-Louis Lassez (1987b). From unification to constraints. In *Proceedings of the SLPT Workshop*, pages 543–560, Lannion. CNET.

Howard Lasnik and Juan Uriagereka (1988). *A Course in GB Syntax: Lectures on Binding and Empty Categories*. MIT Press, Cambridge, Mass.

A. K. Mackworth (1987). Constraint satisfaction. In Stuart Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 205–211. Wiley-Interscience Publication, New-York.

Patrick Saint-Dizier (1988). Parsing natural language with discontinuous grammars. In K. Fuchi and L. Kott, editors, *Programming of Future Generation Computers II*, pages 285–306. Elsevier Science Publishers, North Holland.

Peter Sells (1985). *Lectures on Contemporary Syntactic Theories*. University of Chicago Press, Chicago.

Tim Stowell (1981). *Origins of Phrase Structure*. PhD thesis, MIT, Cambridge, Mass.

Pascal Van Hentenrick (1989). *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, Mass.