Constraint Logic Programming for Natural Language Processing

Nabil Hathout & Patrick Saint-Dizier
IRIT – Université Paul Sabatier
118, route de Narbonne. F-31062 Toulouse Cedex.
e-mail: hathout@irit.fr & stdizier@irit.fr

Abstract

This paper presents some uses of constraint logic programming in the field of natural language processing. It consists of three parts. The first part introduces constraint logic programming. The second one gives a rapid survey of a variety of constraints that may be used to design natural language systems. In the last part, we present the use of the finite domain constraints to model a GB module, namely Binding Theory.

1 Introduction

By way of introduction, let us make the following remarks about the needs of the natural language systems. It is worth noticing that:

- Current natural language systems involve complex feature structure treatments which are based on rewriting and unification.
- Syntactic features and syntactic processes are subject to various constraints. These constraints must be dealt with so that to guarantee their satisfiability throughout the whole process (i.e. the whole parsing or generation process).
- The systems which perform the feature structure treatments must offer a great flexibility in the specification of the syntactic constraints in order that independent aspects of the linguistic theories can be dealt with independently.
- Various linguistic processes do not apply at grammar rule level, but involve larger portions of the parse tree. These processes must be dealt with by means of constraints which are global to the whole grammar, and not local to a single rule.
- The natural language processing tools must preserve the adequacy, the expressiveness and the explanatory power of the linguistic systems they model.

In this paper, we propose some solutions to these points based on constraint logic programming. The main interest of this approach is that it offers a global rule-based framework to handle constraints.

2 Constraint Logic Programming

Constraint logic programming (hereafter CLP) results from the embedding of constraint solving techniques into logic programming (hereafter LP).

2.1 CLP Programs

A CLP program is a set of constrained Horn clauses of the following form:

$$A \leftarrow B_1, \ldots, B_n \quad \{C_1, \ldots, C_m\}$$

where $\{C_1, \ldots, C_m\}$ is a set of constraints called the *constraint system* of the rule. The meaning of this rule is that A can be rewritten as $B_1, \ldots, B_n$ provided that the constraints $C_1, \ldots, C_m$ are simultaneously satisfiable. In other words, A can be rewritten as $B_1, \ldots, B_n$ if there exists at least one instanciation of ground terms for all the variables of the rule which makes the constraints $C_1, \ldots, C_m$ satisfied.

A CLP goal is a constrained formula of the form:

$$\leftarrow G_1, \ldots, G_p \quad \{C_1, \ldots, C_q\}$$

In other words, it is a CLP clause with an empty head. From a procedural point of view, the refutation of a goal is performed by means of the following abstract machine proposed by Alain Colmerauer (1990). This machine has a single non-deterministic transition which can be described by three expressions:

(1) a. $(W, (A_1, \ldots, A_i, \ldots, A_n), S)$

b. $B \leftarrow B_1, \ldots, B_n \quad R$

c. $(W, (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_n), S \cup R \cup \{A_i = B\})$

(1a) represents the current state of the machine. In this formula, W is the set of the variables the values of which we are interested in, that is the set of the variables that occur in the initial query. $A_1, \ldots, A_i, \ldots, A_n$ are the atoms of the current goal; we assume that A_i is the selected atom. S is the current constraint system; it is assumed to be satisfiable. (1b) represents the selected rule of the program, the constraint system of which is R. And (1c) stands for the next state of the machine.

The machine is allowed to go from (1a) to (1c) only if the new constraint system $S \cup R \cup \{A_i = B\}$ is satisfiable. Notice that the unification of the selected atom A_i with the head B of the rule is regarded as a mere identity constraint added to the new constraint system. Thus, this equation replaces the usual substitution of the standard Prolog abstract machine. In addition to the checking of its satisfiability, the new constraint system is also simplified. In particular, every variable that the constraints enforce to have a single value is instanciated (with this value).

2.2 CLP Contributions

Constraint logic programming thus associates unification, Robinson's resolution and constraint solving. It enhances logic programming in many aspects.

First, CLP improves the efficiency of the logic systems because of its active use of the constraints. Actually, the CLP systems maintain the constraints *active* throughout the whole

proof construction process, until they can be adequately solved. The constraints are said to be active in the extent that the satisfiability of the constraint system is checked and the constraint system is simplified at each resolution step. The constraints are actually solved as soon as a sufficient knowledge about their variables is available. Finally, as soon as the constraint system become inconsistent, a backtracking happens.

The use of constraints also improves LP expressiveness because constraints have a different and more general interpretation (Jaffar and Lassez 1987a, Dincbas *et al.* 1988) than that of Prolog predicates. This results form the introduction of *new domains of computation* such as Boolean and arithmetic domains beside the usual Herbrand's one. The user can thus handle directly the objects of the intended domain, namely of the domain of discourse, as opposed to having them encoded as Prolog terms. Similarly, the constraints directly describe the privileged properties of the intended domain, which results in a saving of naturalness. Also, we can by means of constraints represent properties implicitly and describe objects intentionally. We then have available not only the objects, but also the properties that characterize them.

Another major aspect of constraints is that they are fully declarative. They are fully independent of the way they are used, that is to parse or to generate sentences, with a top-down or a bottom-up strategy. Also, they can be stated at any time contrarily to Prolog constraints such as arithmetic predicates, which can only be invoked if certain of their parameters are completely known (Cohen 1990).

Constraints also introduce a greater modularity because each constraint is dealt with by a specific constraint solver which is integrated at unification level. This results in an improvement of the genericity and the reusability of the CLP tools which can be used for different purposes.

3 Constraints for Natural Language Processing

The next part of the paper presents classes of constraints which could be used to write grammars and to design natural language systems.

3.1 Constraints on Terminal Strings

The three following constraints bear on the terminal strings. They can be used to enhance the logic grammar expressiveness and declarativeness. These constraints have to be associated with the grammar rules as constraint systems. Notice that even if they describe relations that hold between items that belong to a single rule, they are global since they constrain the whole syntactic structure. The reason is that a syntactic structure may be produced only if it satisfies all the constraints of the grammar rules used to derive it.

3.1.1 The Precedence Constraint

The most general and basic of the constraints on terminal strings is the precedence one (Saint-Dizier 1991). It merely expresses the linear precedence relation and has the following form: *precedes*(A, B). This constraint states that the part of the syntactic structure derived from A must linearly precede the one derived from B. We can thus write grammars in which all the precedence relations are explicitly stated. Their rules may have the following form:

$$\mathrm{X} \rightarrow \mathrm{Y}_1, \ldots, \mathrm{Y}_n \ \{ \ \mathrm{precedes}(\mathrm{Y}_{i_1},\mathrm{Y}_{j_1}), \ldots, \mathrm{precedes}(\mathrm{Y}_{i_m}, \mathrm{Y}_{j_m})\}$$

Such a rule states that X could, for instance, be derived as the *unordered* list $[Y_1, \ldots, Y_n]$ (which represents the set $\{Y_1, \ldots, Y_n\}$) provided that all the precedence constraints $precedes(Y_{i_1}, Y_{j_1})$, ..., $precedes(Y_{i_m}, Y_{j_m})$ are satisfied. Thus, linguists can write grammars in which the precedence relations are specified only when necessary. This can be interesting, for instance, to describe grammars for free phrase order languages in a very elegant and economic way. The operational semantics of the precedence constraint has been described in (Saint-Dizier 1990).

In addition to the precedence constraint, it seems interesting to us to add to these grammars two other constraints, namely an immediate precedence constraint and a connectedness one.

3.1.2 The Immediate Precedence Constraint

The immediate precedence constraint has the following form: $immediately\_precedes(A, B)$ and states that the part of the syntactic structure derived from A must precede the one derived from B and that these two parts must be adjacent. We can describe this constraint by means of the precedence constraint in the following manner:

$$\text{immediately\_precedes(A, B)} \iff$$
$$(\text{precedes(A, B)} \wedge (\nexists \text{C: precedes(A, C)} \wedge \text{precedes(C, B)}))$$

The immediate precedence constraint constitutes a tool for writing grammars which is finer than the precedence constraint and thus is, in many cases, more adequate than the latter. Besides, it is also stronger than the latter and thus increases the efficiency of the NL systems which use these grammars (cf. 3.1.1): the stronger the constraints are, the more efficient the system is since the search space pruning is more extensive. We can, for example, use this constraint to state that in English, a transitive verb precedes its direct object complement and is adjacent to it:

```
vp -> v, np      {immediately_precedes(v, np)}
```

3.1.3 The Connectedness Constraint

The second constraint that could be added to the grammars presented in 3.1.1 is a connectedness one. It has the following form $connected(X)$, where X is either a symbol or an unordered list of symbols. This constraint states that the part of the syntactic structure derived from X must be connected, i.e. that a symbol which is not dominated by X, cannot occur between symbols (according to the linear precedence relation) dominated by X. The connectedness constraint can be used, for example, to state that a phrase must be connected. This can be described by associating the constraint `connected(XP)` with the rules which describe the phrasal elements.

We can thus see that the precedence constraint is a very basic one, and that several others may be defined on top of it. Moreover, these constraints form a tool which makes the grammars more expressive and also more declarative since precedence, adjacency, and connectedness relations are explicitly stated.

3.2 Arithmetic and Boolean Constraints

Arithmetic and Boolean constraints are the constraints the more commonly found in the CLP systems (Colmerauer 1990, Dincbas *et al.* 1988, Jaffar and Lassez 1987b). The arithmetic

constraints describe relations (i.e. equations, inequalities...) between arithmetic typed terms. Usually, only linear arithmetic constraints are dealt with by CLP systems. Linear arithmetic constraints seem sufficient since natural language processing do not involve the resolution of very intricate arithmetic equations. These constraints are useful, especially for semantic processing and to describe lexical semantics properties. Linear arithmetic constraints are generally solved with linear programming tools. The arithmetic solvers of almost all the CLP systems are based on Simplex-like algorithms. Yet, CLP($\mathbb{R}$), designed by Jaffar and Lassez (1987a &1987b), also deals with non-linear constraints thanks to a delay device which freezes (i.e. delays) the resolution of the non-linear constraints until they become linear.

Other constraints which may help us to design NL systems are Boolean constraints. They may be useful, mainly for feature manipulation, for instance in the manner proposed by Franz Günthner (1988). We can, for instance, compute the gender of a conjunction of NPs in French as follows. Let us choose the feature *female* as gender feature (we may have chosen the feature *male* as well). We then have:

```
np(F) -> np(F1), [et], np(F2)      {F = F1 & F2 }
```

3.3 Long Distance Dependencies: The Pending Constraint

Another class of constraints of much interest for syntactic processing, but also for many other types of processing is the expression of the long-distance relations between constituents in a structure (syntactic, semantic, transfer...). The notion of long-distance dependency will be here formulated as a co-occurrence constraint. This constraint emerged from the logic programming language Dislog presented in (Saint-Dizier 1988, Saint-Dizier 1989). Let us present it briefly.

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

$$\{f_1, \ldots, f_n\}$$

The informal meaning of a Dislog clause is: *if a clause f_i in a Dislog clause is used to construct a given proof tree, then all the other f_js of that Dislog clause must be used to construct that proof tree, with the same substitutions applied to identical variables.* Moreover, there are no hypothesis made on the location of these clauses in the proof tree. For example, the following Dislog clause is composed of two Prolog facts:

$$\{arc(a, b), arc(e, f)\}$$

This clause means that, in a graph, the use of $arc(a, b)$ is conditional to the use of $arc(e, f)$, and *vice-versa*. If one is looking for a path in a graph, this means that all path going through $arc(a, b)$ will have to go through $arc(e, f)$ and conversely.

A Dislog clause thus permits us to express co-occurrence of clauses in a proof tree. The constraint stating that all identical variables in an instance of a Dislog clause must be substituted for the same terms permits the transfer of arguments values between non-contiguous elements in a very convenient way. A Dislog clause can be subject to various types of restrictions such as: linear precedence conditions on the f_is, modalities of application of some f_is, and specification of bounding domains in which a Dislog clause instance must be fully used.

The co-occurrence of two constituents in a larger one can be expressed by the constraint *pending*(A, B), where A is a grammar rule and B is an unordered list of grammar rules (Saint-Dizier 1991). Informally, this constraint means that A originates the pending of the

rules in B. In other words, A can be used in a derivation, only if somewhere else in the derivation (corresponding to the sentence), all the rules in B are also used with identical substitutions applied to identical variables. Notice that the constraint *pending* does not impose any restriction on the location of the constituents derived by means of the rules of B.

4 The Finite Domain Constraints and their Use in GB-Based NLP

The last class of constraint we would like to present is that of the finite domain constraints i.e. of constraints over variables which range over finite domains. The introduction of these constraints aims at embedding constraint propagation techniques inside logic programming (Van Hentenrick 1989). These techniques are based on the idea of *a priori* pruning. In other words, the constraints are used 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 any solution (Freuder 1978, Mackworth 1987). To illustrate the use of these constraints, let us go over the modeling of a Government and Binding module (Chomsky 1981), namely Binding Theory[1].

Binding Theory consists of three conditions. Condition A states that an anaphora must be bound in its governing category. Anaphors consist of reflexives such as *himself* and reciprocals such as *each other*. The governing category of a noun phrase is a certain domain of the syntactic structure which contains that noun phrase. Finally, we say that X binds Y if and only if X c-commands Y and is co-indexed with it. For instance, in a sentence like:

(2) $*$ John believes Mary's description of himself$_i$

The anaphora *himself* is not bound in its governing category. As a result, the sentence is ungrammatical. On the other hand, in the sentence:

(3) Mary believes John$_i$'s description of himself$_i$

the anaphora is bound by *John*$_i$ and the sentence can thus be grammatical. If we adopt Abney's DP-Analysis (Abney 1987), the noun phrase *John's description of himself* which is the governing category of *himself* would have the following representation. One can easily notice that each of the four elements *John*, *'s*, *description*, *of* could be the wanted binder since they all are dominated by the governing category of *himself* and c-command *himself*. However, we can rule out three of these potential binders because *'s* and *of* do not have morpho-syntactic features, also called ϕ-Features, and because *description* does not have a gender feature. Notice that in (2), the four potential binders are ruled out because *Mary*'s gender does not agree with that of *himself*.

More formally, if x is an anaphora, and if $b_1, \ldots, b_n$ are the potential binders of x (i.e. the phrases that c-command x and are dominated by the governing category of x), then Condition A can be expressed as $bound(x, [b_1, \ldots, b_n])$. This constraint states that x must be bound by one of the members of the list $[b_1, \ldots, b_n]$; $bound(x, [b_1, \ldots, b_n])$ is equivalent to the following conjunction:

$$binder(x, Y) \wedge index(x) = index(Y) \wedge \phi\text{-}F(x) = \phi\text{-}F(Y) \wedge belongs(Y, [b_1, \ldots, b_n])$$

[1] Pianesi (1991) presents another constraint based modeling of Binding Theory. Besides, Fong (1990) proposes a related work about free indexation.

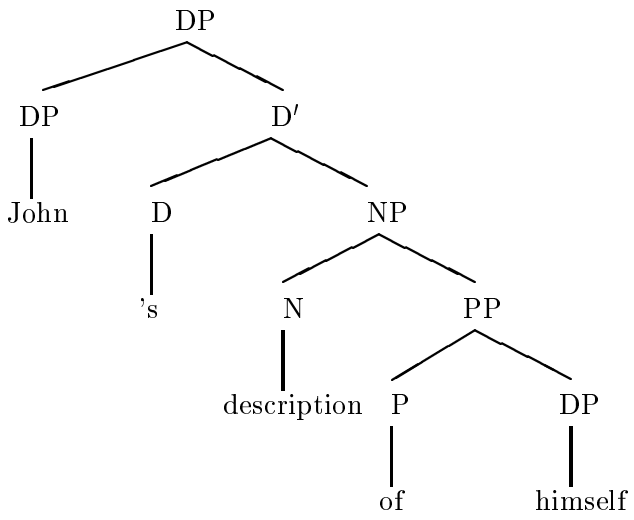

Figure 1: The governing category of *himself*

which states that the binder Y of x must have the same index and the same ϕ-Features as x and must belong to the set of the potential binders of x. In this conjunction, $binder(x, Y)$ is an uninterpreted relation that introduces a new variable Y which represents the binder of x. In this conjunction is the finite domain constraint $belongs(Y, [b_1, \ldots, b_n])$ states that Y must be one of the members of $[b_1, \ldots, b_n]$. The set $\{b_1, \ldots, b_n\}$ is thus the finite domain of the variable Y, and the resolution of the constraint *belongs* consists in removing the members of this domain that cannot have the same index and the same ϕ-Features as x. More generally, we rule out the domain members which cannot be binders of x for one reason or another. During the syntactic process, each time a DP x happens to be an anaphora, we have to compute the set PB of its potential binders and to posit a constraint $bound(x, PB)$.

Binding Theory also comprises two other conditions (Conditions B and C) the modelings of which are essentially identical; we only present one of them here. Condition B states that a pronominal must be free in its governing category. Pronominals are pronouns such as *he* or *him*. The following sentences show the effects of Condition B:

(4) a. $*$ John thinks that $[_{GC}$ Mary$_i$ hates her$_i]$

b. John thinks that $[_{GC}$ Mary$_i$ hates her$_j]$

In the sentence (4a), *her* refers to *Mary* and thus is bound in its governing category which leads to the ungrammaticality of the sentence; we should have had *John think that Mary hates herself*. In the sentence (4b), *her* is not co-referential with *Mary* and the sentence is grammatical. In particular, the pronominal *her* is free in its governing category.

So, if x is a pronominal and if $b_1, \ldots, b_n$ is the set of the potential binders of x, then Condition B can be expressed as $free(x, [b_1, \ldots, b_n])$. This constraint imposes that the index of x must be different from those of the members of $[b_1, \ldots, b_n]$. Each time we find a pronominal, we must posit a constraint *free* corresponding to that noun phrase.

To end this last part, let us notice that the type of modeling we presented for Binding Theory may be used to model almost all the GB principles (Hathout 1991). We only have

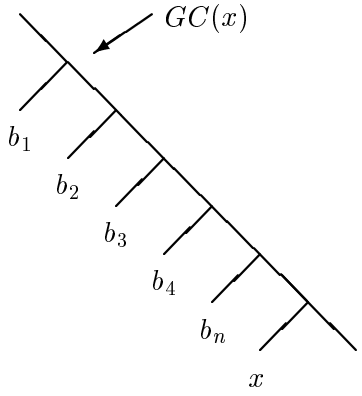

Figure 2: The potential binders of x

to replace the set of the potential binders by that of the potential antecedents, governers, controlers...

5 Conclusion

To summarize, we could say that:

- Constraint logic programming offers a global rule-based framework to handle constraints.
- CLP associates unification, Robinson's resolution and constraint solving.
- The CLP program are fully declarative and soundly based within a unified framework of formal semantics.
- They exhibit a great expressive power and a great ease of use since constraints can be stated at any time.

References

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

Chomsky N. (1981). *Lectures on Government and Binding*, volume 9 of *Studies in Generative Grammar*. Foris, Dordrecht, 1981.

Cohen J. (1990). "Constraint Logic Programming Languages." *Communications of the ACM*, 33(7):52–68, July 1990.

Colmerauer A. (1990). "An Introduction to Prolog III." *Communications of the ACM*, 33(7):69–90, July 1990.

Dincbas M., Van Hentenrick P., Simonis H., Aggoun A., Graf T., and Berthier F. (1988). "The Constraint Logic Programming Language CHIP." In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 693–702, Tokyo, 1988. ICOT.

Fong S. (1990). "Free Indexation: Combinatorial Analysis and A Compositional Algorithm." In *Proceedings of the twenty eighth Annual Meeting of the Association for Computational Linguistics*, pages 105–110, Pittzburg, Pe., 1990.

Freuder E. C. (1978). "Synthetizing Constraint Expressions." *Communications of the ACM*, 21(11):958–966, November 1978.

Günthner F. (1988). "Features and Values 1988." Technical Report SNS-Bericht 88–40, Tübingen University, Tübingen, 1988.

Hathout N. (1991). "Some Aspects of GB-Parsing within the CLP Framework." In *Proceedings of the ICLP'91 Workshop on Advanced Logic Programming Tools and Formalisms for Language Processing*, pages 86–99, Paris, 1991.

Jaffar J. and Lassez J.-L. (1987a). "Constraint Logic Programming." In *Proceedings of the Fourteenth ACM Symposium of the Principles of Programming Languages*, pages 111–119, Munich, 1987. ACM.

Jaffar J. and Lassez J.-L. (1987b). "From Unification to Constraints." In *Proceedings of the SLPT Workshop*, pages 543–560, Lannion, France, 1987. CNET.

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

Pianesi F. (1991). "Indexing and Referential Dependencies within Binding Theory: A Computational Framework." In *Proceedings of the European Chapter of the Association for Computational Linguistics*, pages 39–44, Berlin, 1991. ACL.

Saint-Dizier P. (1988). "Foundations of Dislog, Programming in Logic with Discontinuities." In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1988. ICOT.

Saint-Dizier P. (1989). "Constrained Logic Programming for Natural Language Processing." In *Proceedings of the European Chapter of the Association for Computational Linguistics*, Manchester, 1989.

Saint-Dizier P. (1990). *On Logic Programming Interpretation of Dislog: Programming Discontinuities in Logic.* Lecture Notes in Computer Science. Springer Verlag, October 1990.

Saint-Dizier P. (1991). "Processing Language with Types and Active Constraints." In *Proceedings of the European Chapter of the Association for Computational Linguistics*, Berlin, 1991.

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