A Non-Generative Constraint-Based Formalism

Philippe Blache
LPL-CNRS, Université de Provence
29 Avenue Robert Schuman
13621 Aix-en-Provence, France
pb@lpl.univ-aix.fr

Frank Morawietz
SFS, Universität Tübingen
Wilhelmstr. 113
72074 Tübingen, Germany
frank@sfs.nphil.uni-tuebingen.de

Abstract

Most of modern linguistic theories rely on the notion of constraints. However, only few applications try to implement the parsing process directly with constraint programming: in most cases, constraints are interpreted in a passive sense, the parsing process itself being no different to that of classical derivational approaches. We show in this paper that the problem comes from the representation of linguistic information and the generative interpretation of the relation between grammar and language. We propose a new approach, called *Property Grammar* in which all syntactic information is represented with constraints, the parsing process being an actual constraint resolution mechanism.

1 Introduction

Modern linguistic theories often make use of the notion of *constraint*: constraint-based theories (cf. (Shieber92), (Borsley96)) as well as other paradigms such as principle and parameters (including optimality theory, (Archangeli97)) or dependency grammars (constraints dependency grammars (Maruyama90)) follow this tendency. A constraint approach offers several advantages, in particular a fine-grained representation of information, a clear distinction between linguistic objects and their properties, and a better declarativity.

In computational linguistics, the idea that constraints have a direct counterpart in the implementation is implicit. Unfortunately, this is only partly true and constraints only have local scope during the processing. In spite of several works trying to use constraint programming (see for example (Guenthner88), (Maruyama90) or more recently (Blache96), (Götz97), (Duchier99)), the parsing process cannot be interpreted as actual constraint resolution. This doesn't follow the original idea (see in particular (Sag99)) which proposes the direct use of the linguistic information as a set of constraints over the categories. We show in this paper that this problem has two main origins: the representation of the linguistic information and the influence of the generative conception of the relation between grammars and their generated languages on the linguistic analysis.

We propose a new approach called *Property Grammar* representing all the linguistic information by means of simple constraints. Since these constraints constitute an actual constraint system (in the same sense as a set of equations forms a system), it is then possible to consider parsing as a pure constraint program.

2 Using Constraints

Constraints in linguistic theories generally play only a local role[1]: they are used to control the instantiation of linguistic objects (the categories) and represent relations between these objects (or their components). Then, a clear distinction between the objects and their properties becomes possible, which favors declarativity. For example, in a theory like GPSG, one can find constraints at different levels of the description: linear precedence, feature cooccurrence restriction or universal instantiation principles. But not all the information is represented with constraints and during a parse, one has to select a local tree first and then verify that this tree satisfies the different constraints. This is typically a passive use of constraints.

The problem is different in HPSG in which most of the information is actually represented by means of constraints (cf. (Sag99) and especially for an implementation (Blache96) or (Meurers98)). However, the hierarchical information remains predominant in the sense that verification of constraints can only be applied to an implicit local tree. The verification of the principles for example requires the construction of a hierarchical structure containing a mother and its daughters. This doesn't mean that such a structure has to be fully specified. Some delaying mechanisms can obviously be applied and active constraints can be used locally. But in this case too, we can globally characterize the use of constraints for

[1] Constraints in this case are not active all along the parsing process, but only for specific structures built during the parse. Such constraints are local because their scope is reduced according to the knowledge of the constrained structure.

the parsing process as passive and parsing doesn't amount to pure constraint resolution.

One of the reasons for these problems is the generative interpretation of the relation between grammars and languages. In this case, the notion of derivation is central and parsing an input consists in finding a derivation generating it. Such a conception is revealed in the approaches cited above with the role played by the local trees (which usually amounts to one derivation step). Parsing is then conceived as a derivation mechanism controlled by constraints (often reduced to unification). This certainly stems from the fact that licensing of constraints is implemented as improved versions of the generate-and-test paradigm to ensure some sort of efficiency (and sometimes even termination). So far, we do not know of an implementation which directly uses only constraints to drive the parsing (or the generation) process.

3 Some Desiderata for Constraint-Based Approaches

An optimal use of constraints, both from the linguistic and the computational point of view, should meet some requirements. We present in this section some basic desiderata for a genuine constraint-based approach.

The first important property concerns knowledge representation: all linguistic information has to be represented by means of constraints which form an actual system. This property has an important consequence on the declarativity of the approach. Indeed, it allows a clear distinction between the representation of information (in particular the form of the objects) and the mechanisms for calculating with them.

The second important point is the fact that, insofar as the set of constraints constitutes a system, all the constraints are at the same level. This means that the order of verification of the constraints is not relevant. Obviously, some heuristics can rely on a hierarchization of the constraints or more precisely on the control of their relaxation. But this doesn't alter the fact that the system of constraints as to be taken as a whole. Our approach is thus totally different from the Optimality Theory (see (Archangeli97)) in which constraint ranking contains implicitly linguistic information.

Encapsulation is another important property for constraint-based theories: a constraint must represent homogeneous information. We can then distinguish different types of constraints, each one representing a specific part of linguistic information. Such a requirement avoids implicit information and each type of constraint becomes linguistically motivated.

The last requirement concerns the notion of grammaticality. One interesting property of constraint solving is the possibility to build an approximate solution (when no exact solution can be found) which is nothing but the state of the constraint system after verifying its satisfiability for a given input. As for parsing, finding an exact solution consists in associating a syntactic structure with a given input, e.g. finding a derivation from a distinguished symbol to this input, answering the question of grammaticality: an input is grammatical if a solution can be found. However, the relevant question concerning parsing of real natural language should not be grammaticality, but rather the robustness of providing information about the input. Any kind of input has to be treated. Therefore, we propose to replace the notion of grammaticality with that of *characterization* which is much more general: a characterization is the state of the constraint system for a given input.

4 Property Grammar

Property Grammar (cf. (Blache99), (Bès99)) constitutes an approach implementing these requirements: declarativity, encapsulation, satisfiability. We present in this section their main characteristics. As specified above, all the linguistic information is represented here by means of *properties*.

4.1 The Basic Properties

We use for the representation of syntactic information 7 properties[2] defined as follows :

- **Constituency** (*const*): Set of categories constituting a phrase.
- **Obligation** (*oblig*): Set of compulsory, unique categories (heads).
- **Unicity** (*uniq*): Set of categories which cannot be repeated in a phrase.
- **Requirement** ($\Rightarrow$): Cooccurrence between sets of categories.
- **Exclusion** ($\not\Leftrightarrow$): Restriction of cooccurrence between sets of categories.
- **Linearity** ($\prec$): Linear precedence constraints.
- **Dependency** ($\rightsquigarrow$): Dependency relations between categories.

Let us take some examples (all our examples are from French):

- *Constituency: const(NP) = {Det, N, AP, PP, Sup, Pro}*

  All these constituents can appear within a sequence characterizing a *NP*.

[2]This list of properties can be extended in order to integrate other levels of linguistic analysis such as prosody or semantics.

- *Obligation*: *oblig(NP)* $=\{N, AP, Pro\}$
  In a NP, these categories can be the head.
- *Unicity*: *unic(NP)* $= \{Det, N, AP, PP, Sup, Pro\}$
  Set of unique categories in a *NP*.
- *Linearity*: $Det \prec AP \qquad AP \prec N$
- *Requirement*:
  - $\{le\ \hat{e}tre_{[N,P]}\} \overset{VP}{\Rightarrow} ClR_{[N,P]}$ (Je me le suis dit/*I told that to myself*)
    If the categories *le* (clitic) and a finite verb *être* (with agreement features *N* and *P*) cooccur (characterizing a *VP*), then a reflexive clitic (which agrees with the verb) also has to be present.
- *Exclusion*:
  - $ClR \overset{VP}{\not\Leftrightarrow} lui$ (*Je me lui dis / **I told him myself*)
    In a *VP*, a reflexive clitic cannot cooccur with the clitic *lui*.
- *Dependency*: $Det \leadsto N \qquad AP \leadsto N$

The first important remark is that characterizations are associated with sets of categories. This is interesting in the sense that a characterization can be given independently from the characterization of the non-lexical constituents belonging to this set. The second important point concerns the fact that properties can be expressed over sets of categories, allowing then to represent contextual information. It is the case in particular for the *requirement* and *exclusion* properties, see the indices.

5 Parsing as Constraint Resolution

Insofar as syntactic information forms a constraint system, parsing can then be implemented using constraint programming techniques such as constraint resolution. This approach has several advantages, in particular flexibility and reusability. For example, one can choose to verify only a subset of the constraint system. It is also possible to satisfy constraints with different granularity levels (e.g. a shallow description of the sentence level together with a fine-grained definition of the *NP*). The possibility of integrating several constraints coming from different levels of the linguistic analysis also constitutes an interesting property of using such an approach. But the most important advantage of constraint resolution lies in the fact that it allows a direct interpretation of the notion of characterization.

We present in this section the general architecture of the parsing process. We first describe the elementary trees constituting the basic information and then the constraint resolution process itself.

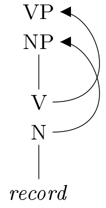
5.1 Elementary trees

Interpreting the parsing process as checking constraint satisfaction can offer a new point of view. Indeed, constraints are applied directly over the objects (the categories) rather than over structures (a local tree). It is thus possible to apply the constraints to the set of all the possible categories that can be associated with the input.

Insofar as constraints are applied over sets of categories (whatever their level, lexical or not), we can associate with each word its possible categories and their possible projections.

Such structures can be represented as an elementary tree as described in (1).

(1)



This example presents the elementary tree of the word *"record"* which can be a noun or a verb. These categories can respectively belong to the *constituent* property of the *NP* and *VP*. The elementary tree is then composed of three nodes: the lexical item, its categorizations and their projections. It is possible to associate such a tree with each lexical item, without any distinction between major and minor categories (i.e. our notion of projection is reduced to the membership of a category to a *const* property).

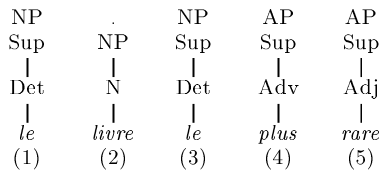
5.2 Satisfying Constraints to Build Characterizations

The parsing process takes as input the set of elementary trees that can be associated with the sentence and builds all the characterizations of this input. A characterization is the state of the constraint system for a particular set of categories, in other words the set of satisfied properties and that of unsatisfied ones.

A *characterization* is then defined over a set of categories by the set P^+ (satisfied properties) and P^- (unsatisfied properties).

Let us take an example. Fig. 1 presents a grammar of the noun phrase in French. It describes in particular the properties of the noun phrase, the adjective phrase and the superlative. In (2), the elementary trees associated with the words of the noun phrase *le livre le plus rare (the rarest book)* are given.

(2)



Properties of the *NP*	
(1)	$Const = \{Det,\ N,\ AP,\ Sup\}$
(2)	$Oblig = \{N,\ AP\}$
(3)	$N[com] \Rightarrow Det$
(4)	$Det \prec N$
(5)	$Det \prec AP$
(6)	$Det \prec Sup$
(8)	$N \prec Sup$
(9)	$AP \not\Leftrightarrow Sup$
(10)	$Det \leadsto N$
(11)	$AP \leadsto N$
(12)	$Sup \leadsto N$

Properties of the *AP*	
(1)	$Const = \{Adj,\ Adv\}$
(2)	$Oblig = \{Adj\}$
(3)	$Adv \prec Adj$
(4)	$Adv \leadsto Adj$

Properties of the *Sup*	
(1)	$Const = \{Det,\ Adv,\ Adj\}$
(2)	$Oblig = \{Adj\}$
(3)	$Adj \Rightarrow Det$
(4)	$Adj \Rightarrow Adv$
(5)	$Det \prec Adv$
(6)	$Det \prec Adv$
(7)	$Adv \prec Adj$
(8)	$Det \leadsto Adj$
(9)	$Adv \leadsto Adj$

Figure 1: A property grammar for the *NP*

The first stage consists in determining the set of categories to be verified. The most general approach consists in characterizing all subsets of categories belonging to the set of elementary trees. Considering all the possible subsets of categories has an exponential cost but some simple control mechanisms can be applied. In the following, we will only take into account the sets of juxtaposed categories (implementing then the projectivity constraint); in this case, sets of constraints are in fact lists.

Building the characterizations consists for each subset of categories of verifying the satisfiability of the properties. The properties that cannot be evaluated remain untouched. This means that a set of properties belonging to a characterization always specifies a phrase-level category. More precisely, it is possible to associate with a set of categories an upper-level category C (or several categories in case of ambiguity). We also say in this case that C is *characterized* by its P^+ and P^- sets.

The following example illustrates the construction of such characterizations for (2) according to the grammar presented in Fig. 1. Each category is associated with its position in the input. The characterization sets follow the sequence of categories with their properties being represented by their indices. For example, the characterization of the set $\{Det_1,\ N_2\}$ indicates that all the properties specified for these categories (i.e. constituency, obligation, requirement, precedence and dependency) are satisfied. The set $\{Adv_4,\ AP_5\}$ presents two characterizations: this sequence can specify an *AP* (all the properties of the grammar are verified) or a *Sup* in which a requirement property is not satisfied.

Note that some characterizations have an empty set of unsatisfied properties (let us call them *positive* characterizations). But no particular status is given to them. This allows the robust treatment of any kind of input, i.e., we do not restrict the analysis to the set of "grammatical" structures.

- Det_1/N_2 :	$P^+(NP) = \{1, 2, 3, 4, 10\}$
	$P^-(NP) = \emptyset$
- Sup_1/N_2 :	$P^+(NP) = \{1, 2, 12\}$
	$P^-(NP) = \{6,3\}$
- Det_3/Sup_4 :	$P^+(NP) = \{1, 6\}$
	$P^-(NP) = \{2\}$
- Det_3/AP_4 :	$P^+(NP) = \{1, 5\}$
	$P^-(NP) = \{2\}$
- Sup_3/AP_4 :	$P^+(NP) = \{1\}$
	$P^-(NP) = \{2, 9\}$
- Adv_4/AP_5 :	$P^+(AP) = \{1, 2, 3, 4\}$
	$P^-(AP) = \emptyset$
	$P^+(Sup) = \{1, 2, 4, 9\}$
	$P^-(Sup) = \{3\}$
- $Det_1/N_2/Sup_3$:	$P^+(NP) = \{1\text{-}4, 6, 8, 9, 10, 12\}$
	$P^-(NP) = \emptyset$
- $Sup_1/N_2/Det_3$:	$P^+(NP) = \{1, 2, 3, 9, 10, 12\}$
	$P^-(NP) = \{4, 6, 8\}$
- $Det_3/Adv_4/Adj_5$:	$P^+(Sup) = \{1, 2, 3, 4, 5, 6, 8, 9\}$
	$P^-(NP) = \emptyset$
- $Det_3/AP_4/Sup_5$:	$P^+(Sup) = \{1, 4, 5, 6\}$
	$P^-(NP) = \{2, 9\}$
- $Det_1/N_2/Sup_3/AP_4$:	$P^+(NP) = \{1\text{-}8, 10, 11, 12\}$
	$P^-(NP) = \{9\}$

Figure 2: Some characterizations of (2)

5.3 Determining Coverage

The core of the process is the construction of the characterization sets. A general mechanism makes use of these sets in order to find a characterization covering the entire input.

A characterization made up of lexical categories (cf. the superlative's characterization in the previous example), constitutes a covering of the segment of the input corresponding to the positions of the constituents. The question is different for embedded constituents. In this case, the set of categories covers the corresponding part of the input if there exists a characterization for all the embedded constituents describing non-lexical categories. In the previous example, the characterization $\{Det_1,\ N_2,\ Sup_3\}$ of the *NP* covers the input because there exists a characterization $\{Det_3,\ Adv_4,\ Adj_5\}$ for its embedded constituent, the *Sup*.

Let us be more precise on the evaluation of covering categorizations. Characterizations don't take into account the status of their embedded constituents. In other words, lexical and non-lexical categories play the same role. This means that each position corresponds to a category. In the case of embedded constituents, we then have several successive categories with the same type as described in (3).

	NP	AP	AP	
	Sup	Sup	Sup	NP
(3)	Det	Adv	Adj	N
	un	*très*	*vieux*	*livre*
	(1)	(2)	(3)	(4)

In this example, one covering of the input will be the set of categories: $\{Det_1,\ AP_2,\ AP_3,\ N_4\}$ which characterizes a noun phrase. The condition for this set to be an actual covering is the presence of a characterization of the AP (which will be in this case the set of categories $\{Adv_2,\ Adj_3\}$).

5.4 Controls and Heuristics

A naive implementation of these mechanisms is obviously not efficient. Indeed, let m be the number of words of the input, c the maximum number of categorizations for each word, then the number n of categories to analyze is bounded by $n = 2mc$. Then, the number of sets of categories to analyze is bounded by 2^n.

Several mechanisms can be added in order to reduce the number of sets to analyze and control the resolution process. In the first case, we can choose to build only sets of juxtaposed categories. At a higher level, one can consider only sets corresponding to subsets of properties (i.e. filtering the sets using the *const* property for example). The satisfaction process itself can be controlled using several heuristics. In particular, it is possible to filter the satisfiability according to a threshold given by the cardinality of the set of unsatisfied properties P^-: it is possible to build only characterizations with less than a certain amount of unsatisfied constraints. At the extreme, we can reduce the satisfiability to positive characterizations.

Another kind of heuristic consists in defining a hierarchization within the set of properties. In this case, the technique of the threshold mentioned above can be modulated with such a hierarchization, some constraints playing a more important role than others.

6 Implementing Property Grammars

Constraint programming is an elegant, declarative way of solving problems with higher-level programming languages. It represents a generalization of logic programming in the sense that it allows the (logical) specification of a problem on an arbitrary domain and tries to solve it with a combination of inference and search. A complete introduction to constraint programming can be found for example in (Marriott98).

We describe in this section an implementation of Property Grammars in a particular constraint programming paradigm. We present this paradigm, Constraint Handling Rules (CHR), in a first step and then explain how to take advantage of it for implementing the resolution process. We want to show with such an implementation that property grammar can be implemented directly using only constraint resolution, which is not actually possible with classical generative approaches.

6.1 Constraint Handling Rules

There are several constraint programming environments available. The most recent and maybe the most flexible is the Constraint Handling Rules package included in SICStus Prolog (ISL95; Frühwirth98). All of these systems provide mechanisms for solving constraints. They maintain a constraint base or store which is continually monitored for possible rule applications, i.e., whether there is enough information present to successfully use a rule to simplify constraints or to derive new constraints. Whereas usually one deals with a fixed constraint domain and a specialized solver, CHR is an extension of the Prolog language which allows for the specification of user-defined constraints and arbitrary solvers. The strength of the CHR approach lies in the fact that it allows for multiple (conjunctively interpreted) heads in rules, that it is flexible and that it is tightly and transparently integrated into the Prolog engine.

In CHR constraints are just distinguished sets of (atomic) formulas. CHR allow the definition of rule sets for constraint solving with three types of rules: Firstly simplification rules (<=>) which replace a number of constraints in the store with new constraints; secondly propagation rules (==>) which add new constraints to the store in case a number of constraints is already present; and thirdly "simpagation" rules (<=> in combination with a \ in the head of the rule) which replace only those constraints with new ones which are to the right of the backslash.

To improve efficiency, rules can have guards and pragma declarations. A guard (separated from the rest of the body by a |) is a condition which has to be met before the rule can be applied. A pragma declaration affects the way the rule is compiled. Our approach will only introduce the *passive* declaration, so we ignore the other possibilities. Declaring a constraint in the head passive simply means that this constraint will not trigger the rule on its appearance in the constraint store.

We cannot go into the details of the formal semantics of CHR here. The interested reader is referred to (Frühwirth98). Let us just note that logically, simplification rules are equivalences and propagation rules are implications if their guard is satisfied. Simpagation rules are special cases of simplification rules. Soundness and completeness results for CHR are available (Abdennadher96; Abdennadher98).

```
prec @ cat(X,N)#I1, cat(Z,N)#I2, cat(Y,M)#I3, cat(Z,M)#I4, prec(Z,Cat1,Cat2,Id) ==>
       N < M |
       ( ( X == Cat2, Y == Cat1 ) ->
             pminus(Z-Id,prec,X-N,Y-M)
       ;
             pplus(Z-Id,prec,X-N,Y-M) )
       pragma passive(I1), passive(I2), passive(I3), passive(I4).
```

Figure 3: An example CHR rule

6.2 The Implementation in CHR

For the implementation in CHR, we simply have to declare the basic properties as constraints, see Sec. 4. Additionally, we use constraints for the sets of satisfied/nonsatisfied constraints, i.e., we have constraints `pplus/4` and `pminus/4` which encode in their first argument the set of constraints we are considering and an identifier, in their second argument the type of constraint and in the last two the categories involved.

The program takes the input string, a Prolog list of words, and traverses it.[3] For each lexical entry, it posts the corresponding category constraint, i.e., `cat(X,Y)` where `X` is a category and `Y` its position in the numeration.

The CHR implementation immediately starts out by building the elementary trees of the initial categories from the posted `cat/2` constraints via constraint propagation. In general, the CHR engine, i.e., all constraint resolution steps, is triggered as soon as constraints matching the heads are posted.

Basically, the CHR program looks at two category constraints and – if they are in the set of constraints we are considering– uses them together with a constraint from the grammar, as the head of constraint propagation rules. The constraints propagated contain the information whether the pair of categories satisfy the constraint from the grammar or not. In this way, we build a complete table of results on all pairs of categories.[4]

So, we build a complete table of `pplus/4`, `pminus/4` constraints for all pairs of categories matching the heads of our propagation rules. These constraints store the information about the interaction of any two categories with the resulting rule and set of constraints which licensed them. Nothing more has to be implemented to achieve this behaviour since it is built into the CHR engine. All we have to do is specify the right propagation rules and to post the constraints from the grammar and input which makes for a very simple and easy to understand program.

Generally, in all the rules dealing with the constraints from the grammar, we apply the rule if we can find two categories within the constraint store which are in the same constituent and a constraint which checks some configuration on them. Only this last constraint triggers the rule since we declared all other constraints to be passive. Note that this ensures that we did all possible projections in each constituent before we post other constraints.

Since, if two category constraints match the head of a rule, they do so in either order, we use the guard to force the linear order as it appears in the input string.

An example of such a propagation rule is given in Fig. 3. We are considering a precedence constraint `Id` on the categories `X` and `Y` within the constituent `Z` with indices `N` and `M`. We check satisfiability of the constraint in the head of the implication and then add the corresponding `pplus` or `pminus` constraint to the store.[5] Note that all the constraints actually appearing in the constraint store are ground such that we can test for literal identity.

The post-processing then just collects all the sets, gets all pairs of appearing categories if they are within the same range and collects all positive and negative constraints attached to them. While doing so, we interpret its members in the following way:

- if we have a successful obligatory or implicational constraint on any two categories within a constituent, we can ignore all the failed ones;
- if we have a failed precedence or co-occurrence constraint on any two categories within a constituent, we have to ignore all the successful ones.

Furthermore, the output of the program – consisting of the generated `pplus`/`pminus`-sets – can be varied according to the given threshold of failed constraints,

[3]Currently from left-to-right although this is arbitrary and can be varied.

[4]Since in the end we are interested in bigger chunks of lexical items then just two, we have to do some post-processing to interpret the entries in the table in the right way. For example, if a precedence constraint succeeds on two pairs of categories, this does not imply that it also succeeds for the entire triple. So, one violation among all the pairs is sufficient to cause failure of that constraint.

[5]We need four `cat` constraints to ensure that both categories belong to the same constituent, i.e., that `Z` indeed covers `N` and `M`.

in the extreme case reducing the process to finding only the positive characterizations.

7 Conclusion

The representation of linguistic information by means of constraints is interesting both for knowledge representation (no implicit information, encapsulation) and from an implementational point of view as well (parsing is seen as constraint solving). Property Grammar offers a framework taking advantage of these characteristics and presenting several interesting insights for natural language processing. The first one concerns the generality and the robustness of the approach: constraints allow the introduction of the notion of characterization to replace that of grammaticality. The parsing process consists in verifying constraints for the possible set of categories associated with the input instead of trying to build a structure according to the grammar. One other interesting point concerns the integration of different information sources: the properties can concern all kind of information (prosodic, syntactic, pragmatic etc.).

A possible direct application of Property Grammars would be the development of tools to work with corpora. Since the modular and flexible nature of these simple constraints is close to the descriptions used by empirical linguists, it could be used to test or find examples of particular grammatical constructions (or even violations of principles) easily by focusing the analysis and using a shallow analysis for the rest. But obviously this would presuppose the development of a dedicated, more efficient constraint solver first.

References

Abdennadher S., T. Frühwirth & H. Meuss (1996) "On Confluence of Constraint Handling Rules", in *Lecture Notes in Computer Science*, vol. 1118.

Abdennadher S. (1998) *Analyse von regelbasierten Constraintlösern*, Ph.D. Thesis, Ludwig-Maximilians-Universität München.

Archangeli D. & D.T. Langendoen eds. (1997) *Optimality Theory*, Blackwell.

Bès G. & P. Blache (1999) *Propriétés et analyse d'un langage*, in actes de TALN'99.

Blache P. & J.-L. Paquelin (1996) *Active Constraints for a Direct Interpretation of HPSG*, in proceedings of HPSG'96.

Blache P. (1999) *Filtering and Fusion: A Technique for Parsing with Properties*, in proceedings of NLPRS'99.

Borsley R. (1996), *Modern Phrase Structure Grammars*, Blackwell.

Carpenter B. & Penn G. (1995) "Compiling Typed Attribute-Value Logic Grammars", in H. Bunt and M. Tomita (eds.), *Current Issues in Parsing Technologies*, Kluwer.

Duchier D. & Thater S. (1999) "Parsing with Tree Descriptions: a constraint based approach", in proceedings of *NLULP'99*.

Frühwirth T. (1998) "Theory and Practice of Constraint Handling Rules", in *Journal of Logic Programming*, 37:1-3.

Götz T., D. Meurers & D. Gerdemann (1997), *The ConTroll Manual*, SFS Report.

Guenthner F. (1988) "Features and Values 1988", *CIS-Bericht-90-2*, Universität München.

Intelligent Systems Laboratory (1995), *SICStus Prolog User's Manual*, SICS.

Marriott, K. & Stuckey, P. J. (1998) *Programming with Constraints*, MIT Press.

Maruyama H. (1990), "Structural Disambiguation with Constraint Propagation", in proceedings of *ACL'90*.

Meurers D. & Minnen G. (1998) "Off-line Constraint Propagation for Efficient HPSG Processing", in G. Webelhuth, J.-P. Koenig, A. Kathol (eds.): *Lexical and Constructional Aspects of Linguistic Explanation*, CSLI.

Morawietz F. & Cornell T. (1997) "Representing Constraints with Automata", in proceedings of the ACL/EACL.

Sag I. & T. Wasow (1999), *Syntactic Theory. A Formal Introduction*, CSLI.

Saraswat V. (1993), *Concurrent Constraint Programming*, MIT Press.

Shieber S. (1992) *Constraint-Based Grammar Formalisms*, MIT Press.