Numbat: Abolishing Privileges when Licensing New Constituents in Constraint-Oriented Parsing

Jean-Philippe Prost

Numbat: Abolishing Privileges when Licensing New Constituents in Constraint-oriented Parsing

Jean-Philippe Prost
Centre for Language Technology
Macquarie University, Sydney, Australia
and *Laboratoire Parole et Langage*
Université de Provence, Aix-en-Provence, France
jpprost@ics.mq.edu.au

Abstract

The constraint-oriented approaches to language processing step back from the generative theory and make it possible, in theory, to deal with all types of linguistic relationships (e.g. dependency, linear precedence or immediate dominance) with the same importance when parsing an input utterance. Yet in practice, all implemented constraint-oriented parsing strategies still need to discriminate between "important" and "not-so-important" types of relations during the parsing process.

In this paper we introduce a new constraint-oriented parsing strategy based on *Property Grammars*, which overcomes this drawback and grants the same importance to all types of relations.

1 Introduction

In linguistics, the term *gradience* is often used to refer to the notion of acceptability as a gradient, as opposed to a more classical all-or-none notion. The research goal of this project is to build an experimental platform for computing gradience, i.e. for quantifying the degree of acceptability of an input utterance. We called this platform Numbat.

In order to be able to quantify such a gradient of acceptability with no *a priori* opinion on the influence played by different types of linguistic relationships, we want to adopt a framework where no one type of (syntactic) relation (e.g. dependency, immediate dominance, or linear precedence) is preferred over the other ones. Although a constraint-oriented (CO) paradigm such as *Property Grammars* (Blache, 2001) theoretically does not rely on any preferred relations, we observe that the parsing strategies implemented so far (Morawietz and Blache, 2002; Balfourier et al., 2002; Dahl and Blache, 2004; VanRullen, 2005) do not account for such a feature of the formalism. The strategy we have designed overcomes that problem and allows for constituents to be licensed by any type of relation. Not only does our approach maintain a close connection between implementation and underpinning theory, but it also allows for the decisions made with respect to gradience to be better informed. The purpose of the present paper is to present this new parsing strategy, and to emphasise how it "abolishes the privilege" usually only granted to a subset of syntactic relationships.

Section 2 presents some background information about the CO approaches and briefly introduces the Property Grammars formalism. Section 3 exposes and discusses the parsing strategy implemented in Numbat. Section 4 then draws the conclusion.

2 Constraint-oriented Approaches

The main feature common to all Constraint-oriented approaches is that parsing is modelled as a Constraint Satisfaction Problem (CSP). Maruyama's Constraint Dependency Grammar (CDG) (Maruyama, 1990) is the first formalism to introduce the parsing process as a CSP solver. Several extensions of CDG have then been proposed (Heinecke et al., 1998; Duchier, 1999; Foth et al., 2004).

Menzel and colleagues (Heinecke et al., 1998; Foth et al., 2004) developed a weighted (or "graded") version of CDG. Their parsing strategies are explored in the context of robust parsing. These strategies are based on an over-generation of candidate solutions. In this approach the CSP is turned into an optimisation problem, where sub-optimal solutions are filtered out according to a function of the weights associated to the violated constraints, and the notion of well-formedness is replaced by one of optimality. Indeed, the over-generation introduces inconsistencies in the constraint system, which prevents the use of the con-

Proceedings of the 3rd Workshop on Constraints and Language Processing (CSLP-06), pages 17–24,
Sydney, July 2006.

straint system as a set of well-formedness conditions, since even a well-formed utterance violates a subset of constraints. Consequently it is not possible to distinguish an optimal structure of an ill-formed utterance from an optimal structure of a well-formed utterance.

Duchier (1999) relies on *set constraints* and *selection constraints*[1] to axiomatise syntactic well-formedness and provides a concurrent constraint programming account of the parsing process. With the *eXtended Dependency Grammar* (XDG) (Debusmann et al., 2004) the notion of dependency tree is further extended to "multi-dimensional" dependency graph, where each dimension (e.g. *Immediate Dominance* and *Linear Precedence*) is associated with its own set of well-formedness conditions (called *principles*). Duchier (2000) sees dependency parsing as a *configuration problem*, where given a finite set of components (nodes in a graph) and a set of constraints specifying how these components may be connected, the task consists of finding a solution tree.

It seems, to the best of our knowledge, that neither of these works around XDG attempts to account for ill-formedness.

The Property Grammars (PG), introduced by Blache (Blache, 2001; Blache, 2005)[2], step back from Dependency Grammar. Solving the constraint system no longer results in a dependency structure but in a phrase structure, whose granularity may be tailored from a shallow one (i.e. a collection of disconnected components) to a deep one (i.e. a single hierarchical structure of constituents) according to application requirements[3]. This feature makes the formalism well suited for accounting for both ill-formedness and well-formedness, which is a key requirement for our experimental platform.

Introducing degrees of acceptability for an utterance does not mean indeed that it should be done at the expense of well-formedness: we want our model to account for ill-formedness and yet to also be able to recognise and acknowledge when an utterance is well-formed. This requirement rules out Optimality-theoretic frameworks as well as the ones based on Maruyama's CDG. Note that this is not to say that the task could not be achieved in a CDG-based framework; simply at this stage there is no work based on CDG, which would combine both an account of well-formedness and of optimality. A CO framework based on PG seems therefore best-suited for our purpose. Meanwhile, though different parsing strategies have been proposed for PG (Morawietz and Blache, 2002; Balfourier et al., 2002; Dahl and Blache, 2004; VanRullen, 2005), none of these strategies implements the possibility afforded by the theory to rely on *any* type of constraint in order to license a (possibly ill-formed) constituent.

We will see in this paper how the parsing strategy implemented in Numbat overcomes this problem.

2.1 The *Property Grammars* Formalism

2.1.1 Terminology

Construction. In PG a *construction* can be a lexical item's Part-of-Speech, a phrase, or top-level constructions such as, for example, the Caused-motion or the Subject-auxiliary Inversion constructions. The notion of construction is similar to the one in Construction Grammar (CxG)[4], as in (Goldberg, 1995), where:

> Cx is a construction *iff* Cx is a form-meaning pair $\langle F_i, S_i \rangle$ such that some aspect of F_i or some aspect of S_i is not strictly predictable from Cx's component parts or from other previously established constructions.

In this paper we only focus on syntax. For us, at the syntactic level, a construction is defined by a *form*, where a form is specified as a list of properties. When building a traditional phrase structure (i.e. a hierarchical structure of *constituents*) a construction can be simply seen as a non-terminal.

Property. A *property* is a constraint, which models a relationship among constructions. PG pre-defines several types of properties, which are specified according to their semantics. Moreover, the framework allows for new types to be defined.

[1]Although they are referred to with the same name by their respective authors, Duchier's notion of *selection constraint* is not to be confused with Dahl's *selection constraints* (Dahl and Blache, 2004). The two notions are significantly different.

[2]The Property Grammars were defined on the basis of the 5P formalism (Bès and Blache, 1999).

[3]For a discussion regarding PG and parsing with variable granularity see (VanRullen, 2005).

[4]Blache (2004) discussed how PG can be used as a formal framework for CxG.

In Numbat, a property type is also called a *relation*. Section 2.1.2 briefly presents some of the pre-defined property types and their semantics.

Assignment. In PG an *assignment* is a list of constituents. Let's consider, for example, the three constituents DET, ADJ and N, the following lists are possible assignments: [DET], [ADJ], [DET, ADJ], [ADJ, N], [DET, N], [DET, ADJ, N], etc..

2.1.2 Some Pre-defined Property Types

Here are some property types pre-defined in PG. See (Blache, 2005) for more types and more detailed definitions.

Notation. We note:

- $\mathcal{K}$ a set of constructions, with $\{\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2\} \in \mathcal{K}$;
- $\mathbb{C}$ a set of constituents, with $\{c, c_1, c_2\} \in \mathbb{C}$;
- $\mathcal{A}$ an assignment;
- ind a function such that $ind(c, \mathcal{A})$ is the index of c in $\mathcal{A}$;
- cx a function such that $cx(c)$ is the construction of c;
- $\mathcal{P}(\mathcal{C}_1,\ \mathcal{C}_2)[c_1, c_2, \mathcal{A}]$ or $(\mathcal{C}_1\ \mathcal{P}\ \mathcal{C}_2)[c_1, c_2, \mathcal{A}]$ the constraint such that the relation $\mathcal{P}$ parametered with $(\mathcal{C}_1, \mathcal{C}_2)$, applies to $[c_1, c_2, \mathcal{A}]$.

Linear Precedence ($\prec$).
By definition, $(\mathcal{C}_1\ \prec\ \mathcal{C}_2)[c_1, c_2, \mathcal{A}]$ holds iff

$$\begin{cases} cx(c_1) = \mathcal{C}_1, & \text{and} \\ cx(c_2) = \mathcal{C}_2, & \text{and} \\ \{c_1, c_2\} \in \mathcal{A}, & \text{and} \\ ind(c_1, \mathcal{A}) < ind(c_2, \mathcal{A}) & \end{cases}$$

Exclusion ($\not\Leftrightarrow$).
By definition, $(\mathcal{C}_1\ \not\Leftrightarrow\ \mathcal{C}_2)[c_1, c_2, \mathcal{A}]$ holds iff

$$\begin{cases} cx(c_1) = \mathcal{C}_1, & \text{and} \\ cx(c_2) = \mathcal{C}_2, & \text{and} \\ \{c_1, c_2\} \cap \mathcal{A} \neq \{c_1, c_2\} & \end{cases}$$

Uniqueness ($Uniq$).
By definition, $Uniq(\mathcal{C})[c, \mathcal{A}]$ holds iff

$$\begin{cases} cx(c) = \mathcal{C}, & \text{and} \\ c \in \mathcal{A}, & \text{and} \\ \forall c' \in \mathcal{A} \backslash \{c\}, cx(c') \neq \mathcal{C} & \end{cases}$$

2.2 Related Problems

CO parsing with PG is an intersection of different classes of constraint-related problems, each of which is listed below.

Configuration problem. Given a set of components and a set of constraints specifying how these components can be connected, a configuration problem consists of finding a solution tree which connects the components together. Deep parsing with PG is a configuration problem where the components are constituents, and the resulting structure is a phrase structure. By extension, a solution to such a problem is called a *configuration*. A configuration problem can be modelled with a (static) CSP.

Dynamic CSP. In our case the problem is actually dynamic, in that the set of constraints to be solved evolves by the addition of new constraints. As we will see it later new constituents are inferred during the parsing process, and subsequently new constraints are dynamically added to the system. When dealing with deep parsing, i.e. with well-formedness only, the problem can be tackled as a *Dynamic CSP*, and solving techniques such as *Local Search* (Verfaillie and Schiex, 1994) can be applied.

Optimisation problem. In order to account for ill-formedness as well as well-formedness, we need to allow constraint relaxation, which turns the problem into an optimisation one. The expected outcome is thus an optimal configuration with respect to some valuation function. Should the input be well-formed, no constraints are relaxed and the expected outcome is a full parse. Should the input be ill-formed, constraints are relaxed and the expected outcome is either an optimal full parse or a set of (optimal) partial parses.

3 Numbat Architecture

3.1 The Parsing Strategy in Numbat

Relying on a design pattern used in various optimisation techniques, such as *dynamic programming*, the top-level strategy adopted in Numbat consists in three main steps:

1. splitting the problem into overlapping sub-problems;
2. solving the sub-problems—or building optimal sub-solutions;

3. building an optimal global solution, using the sub-solutions.

More specifically, the strategy adopted proceeds by successive *generate-and-test*: the possible models to local systems are generated, then their satisfiability is tested against the grammar. The partial solutions are re-injected in the process dynamically, and the basic process is iterated again. Note that the generate-and-test method is not compulsory and is only chosen here because it allows us to conveniently control and then filter the assignments.

Given an input utterance, the parsing process is made up of a re-iteration of the basic following steps:

1. **Building Site**. Build a set of constituents;
2. **Assignation**. Build all the possible assignments, i.e. all the possible combinations of one or more constituents;
3. **Checkpoint Alpha**. Filter out illegal assignments;
4. **Appropriation**. For every assignment, identify and build all the relevant properties among its elements, which leaves us with a property store, i.e. a constraint system;
5. **Checkpoint Bravo**. Filter out illegal assignments and irrelevant properties;
6. **Satisfaction**. Solve the constraint system;
7. **Formation**. Identify forms of construction, i.e. subsets of properties from the property store and nominate the corresponding candidate constructions;
8. **Polling booth**. Decide which of the candidate constructions are licensed and carried over to the next iteration;

The process stops when no new constituent can be built.

Each of these steps is defined in the following section.

3.1.1 Building Site

During the first iteration, this phase builds one constituent for each Part-of-Speech (POS) associated with an input word. From the second iteration onwards, new constituents are built provided the candidate assignments output by the previous round.

3.1.2 Assignation

From one iteration to the next new assignments are built, involving at least one of the new constituents. These constituents result from the previous iteration. Notice that the amount of new assignments created by each iteration grows exponentially with the amount of constituents (the 'old' ones and the new ones). Fortunately, the next step will filter out a large proportion of them.

This phase of assignation is essential to the process, and makes Numbat different from any other parsing strategy for PG. The difference will be made clear in the Satisfaction phase.

3.1.3 Checkpoint Alpha

In Numbat we use a *filtering profile* to specify which combination of heuristics applies during the parsing process. This feature proves to be very useful when performing experiments, as it allows an incremental approach, in order to determine the relative importance of each of the criteria on gradience by turning on and off one or other heuristic.

The heuristics play different roles. They are primarily used to prune the search space as early as possible in the process. Meanwhile, most of them capture language specific aspects (e.g. Contiguity, see below). These language specific heuristics are already present in previous works on PG in one form or another. We are working in the same framework and accept these restrictions, which might be relaxed by future work on the formal side.

During Checkpoint Alpha the following heuristics may apply.

Heuristic 1 (Distinct Constituents) *An assignment may contain no pairwise intersecting constituents.*

That is, any two constituents may not have any constituent in common. For example, the constituents $\{\text{DET}1, \text{ADJ}2\}$ and $\{\text{ADJ}2, \text{NOUN}3\}$ may not belong to the same assignment, since they have one constituent in common.

Heuristic 2 (Contiguity) *An assignment is a set of contiguous elements.*

This heuristic rules out crossing-over elements. Although this heuristic has little consequence when dealing with languages such as French or English, it may have to be turned off for languages with cross-serial dependencies such as Dutch. But if turned off, an additional problem then occurs that the semantics of pre-defined property types must be re-defined. The linear precedence, for instance, would need to account for the order between two crossing-over phrases, which is not the case in the current definition. On the other hand, notice that long distance dependencies are *not* ruled out by heuristic 2, since nested constituents are still legal.

3.1.4 Appropriation

This step has to do with the gathering of all the properties relevant to every assignment from the grammar. This operation is made easier by preprocessing the grammar, which is done at an initialisation step. During this preliminary phase, a lookup table is created for the grammar, where all the properties are indexed by their operands. Every property is also linked directly to the constructions for which it participates in the definition— i.e. the constructions for which the property is a member of the form. This table is actually a hash table, where the keys are the constructions on which the properties hold. For example, the property (Det $\prec$ Noun) is indexed by the couple of constructions (Det, Noun). And the property ($\{$Pronoun, Adv$\}$ $\not\Leftrightarrow$ V) is indexed by the triplets of constructions (Pronoun, Adv, V). Thus, given an assignment, i.e. a set of constituents, all we have to do here is to retrieve all the relevant properties from the lookup table, using all the (relevant) combinations of constituents as keys.

3.1.5 Checkpoint Bravo

Filters apply here, which aim to prune again the search space. The following heuristics may apply.

Heuristic 3 (Full Coverage) *Every element of an assignment must be involved in at least one constraint. That is, for each element in an assignment there must be at least one constraint defined over this element.*

Example 1 *Consider the assignment* $\mathcal{A} = \langle Det, N, V\rangle$*, and the grammar made up of the following properties:*

$$\text{VP} ::= \{V \prec \text{NP}\} \quad (1)$$

$$\text{NP} ::= \{Uniq(\text{N}),\ Det \prec \text{N},\ \text{N} \prec Adj\} \quad (2)$$

$$\text{S} ::= \{\text{NP} \prec \text{VP}\} \quad (3)$$

According to heuristic 3 $\mathcal{A}$ *is ruled out, since the* V *element is not* covered *by any constraints, whether we build an NP or a VP.*

Notice that this heuristic is semantically equivalent to the *Constituency* property present in early versions of PG[5]. The *Constituency* property used to specify which types of constituent (i.e. constructions) were legal ones (for a construction). Such a constraint is unnecessary since the information can be retrieved by simply listing all the types of constituents used in the definitions of properties. In example 1 for instance, the set of legal constituents for the NP construction is $[Det, N, Adj]$.

A main reason for dealing with constituency as a filter rather than as a constraint is to improve efficiency by reducing the amount of constraints in the system. Indeed, a filter aims to rule out constraints, which are subsequently removed from the constraint system. If dealt with as a constraint itself, Constituency would only make the constraint system more complex.

Heuristic 3 raises the issue of ruling out assignments with "free" constituents, i.e. constituents which are not connected to the rest of the assignment. Such a situation may occur, for example, in the case of an unknown word, either because it is absent from the lexicon, or misspelled. We choose to leave it up to the grammar writer to design their own *ad hoc* solutions regarding how to handle such cases. It may be done, for instance, through the definition of a "wildcard construction", and perhaps also a "wildcard property type", which will be used appropriately in the grammar.

3.1.6 Satisfaction

At this stage, only legal assignments and relevant properties are kept in the system. All the required information for evaluating the properties is thus available and all we have to do now is to solve the constraint system.

The solver we use is implemented in Constraint Handling Rules (CHR) (Frühwirth, 1994). Unlike other CHR implementations of PG (Morawietz and Blache, 2002; Dahl and Blache, 2004) where the semantics of the property types are encoded in the handlers[6]—and therefore each type of property requires a different handler—, the approach we have adopted allows us to externalise the semantics and to generalise the properties evaluation with one single handler. The algorithm un-

[5] The *Constituency* property is discarded in the version of PG underpinning Numbat.

[6] A CHR handler is a rule of the general form `(A => B | C)`, which can be read "if A then (if B then C)"

derlying this handler can be expressed as follows:

for each (list of n *constituents, assignment, property*)
if (the list of n *constituents* and the *assignment* match the *property*'s ones)
then
if (*property* is satisfied)
then (tick *property* as being SATISFIED)
else (tick *property* as being VIOLATED)

The CHR handler takes the following form:

```
listOfConstituents(Ccs) &&
assignment(Asg) &&
property(Pp) ==>
  Pp.isConsistentWith(Asg,Ccs) |
    (Pp.isSatisfied() ->
      sat(Pp) ; unSat(Pp)).
```

3.1.7 Formation

This phase is concerned with identifying the constructions in the grammar which can be triggered (i.e. licensed) by the properties present in the property store. A construction is *triggered* by any of the properties which are used to define this construction. This task can be performed easily by accessing them directly in the lookup table (see section 3.1.4), using a property's operands as the key. The constructions which are triggered are called *target constructions*. We then build a constituent for each of these target construction. Such a constituent is called a *candidate constituent*.

This phase basically builds constituent structures. During the next iteration these candidates may be used in turn as constituents. The process thus accounts for recursive structures as well as non-recursive ones. Meanwhile, it is interesting to emphasise that building such a constituent structure is not necessary when parsing with PG. We could, for instance, deal with the whole sentence at once as a sequence of word order constraints. This way no constituent structure would be needed to license infinite sets of strings. In this case, the efficiency of such a process is something that has been worked on extensively within the CSP field. What we are contributing is merely a representation and translation to CSP, which allows us to take advantage of these efficiencies that decades of other work have produced.

Monotonic and Non-monotonic Constraints. The notions of *Selection Constraint* in (Dahl and Blache, 2004) and of *non-Lacunar Constraint* in (VanRullen, 2005) are equivalent and denote a class of constraint types, whose semantics is monotonic, in that their satisfiability does not change when new elements are added to the assignment. Constraint types such as Linear Precedence or Obligation, for example, are monotonic. On the other hand the constraint $Uniq(\mathcal{C})[c, \mathcal{A}]$ (see 2.1.2), for example, is non-monotonic: if the contextual assignment $\mathcal{A}$ grows—i.e. if new constituents are added to it—the constraint needs to be re-evaluated. In parsing strategies where the assignments are built dynamically by successive additions of new constituents, the evaluation of the relevant constraints is performed on the fly, which means that the non-monotonic constraints need to be re-evaluated every time the assignment grows. This problem is tackled in different ways, according to implementation. But we observe that in all cases, the decision to trigger new candidate constituents relies only on the evaluation of the monotonic constraints. The decision process usually simply ignores the non-monotonic ones. Numbat, by fixing the assignments prior to evaluating the local constraint systems, includes both the monotonic and the non-monotonic constraints in the licensing process (i.e. in the Formation phase).

3.1.8 Polling Booth

This phase is concerned with the election process, which leads to choosing the candidates who will make it to the next iteration.

The following heuristics may apply.

Heuristic 4 (Minimum Satisfaction) *An assignment is valid only if at least one constraint holds on any of its constituents.*

Notice that in all other implementations of PG this heuristic is much more restrictive and requires that a *monotonic* constraint must hold.

Heuristic 5 (Full Input Span) *A valid (partial or final) solution to the parsing problem is either a single constituent which spans exactly the input utterance, or a combination of constituents (i.e. a combination of partial parses) which spans exactly the input utterance.*

In theory, we want the Polling Booth to build all the candidate constituents we have identified, and re-inject them in the system for new iterations. In practice, different strategies may apply in order to prune the search space, such as strategies based on the use of a ranking function. In our case, every iteration of the parsing process only propagates one valid combination of constituents to the next iteration (e.g. the best one according to a valuation function). Somehow such a strategy corresponds to always providing the main process with a "disambiguated" set of input constituents from one iteration to another. This heuristic may also be used as a termination rule.

A question then arises regarding the relaxation policy: Do all the constraint types carry same importance with respect to relaxation? This question addresses the relative importance of different constraint types with respect to acceptability. Does, for instance, the violation of a constraint of Linear Precedence between a Determiner and a Noun in a Noun Phrase have the same impact on the overall acceptability of the Noun Phrase than the violation of Uniqueness of the Noun (still within a Noun Phrase)? From a linguistic point of view, the answer to that question is not straightforward and requires number of empirical studies. Some works have been carried out (Gibson, 2000; Keller, 2000), which aim to provide elements of answer in very targeted syntactic contexts.

The impact that the relaxation of different constraint types has on acceptability should not be biased by a particular parsing strategy. Thus, the framework provides the linguist (and the grammar writer) with maximum flexibility when it comes to decide the cost of relaxing different types of constraint on acceptability, since *any type* may be relaxed. Intuitively, one can clearly relax (in French) a constraint of Agreement in gender between determiner and noun; on the other hand one could not as easily relax constraints of type Obligation, which are often used to specify heads. A complete breakdown of constraints into relaxable and non-relaxable is future work. But at the end, the parser just produces sets of satisfied and violated constraints, regardless of how important they are. There will then be a separate process for predicting gradience, where the relative importance of particular constraints in determining acceptability will be decided experimentally.

4 Conclusion

In this paper we have presented the constraint-oriented parsing strategy based on Property Grammars, that we have developed as part of the Numbat platform. We have also demonstrated that, unlike other existing parsers for PG, this strategy does not privilege any particular type of property when licensing a new constituent. By doing so, this parser contributes to maintain a close connection with the underpinning theory. In the context of robust parsing, where decisions must be made on the basis of a balance between satisfied and violated properties, it also allows the decision process to be better informed by providing it with more grounding linguistic material concerning the input.

For the same reason, this contribution is also fairly valuable in the context of our prime research goal, which is concerned with quantifying acceptability.

In further works we plan to evaluate the performance of the parser. We also plan to use Numbat to run series of experiments on gradience, in order to design and test a suitable valuation function to be used to assess the degree of acceptability of an input utterance.

References

Jean-Marie Balfourier, Philippe Blache, and Tristan Van Rullen. 2002. From Shallow to Deep Parsing Using Constraint Satisfaction. In *Proc. of the 6th Int'l Conference on Computational Linguistics (COLING 2002)*.

Gabriel Bès and Philippe Blache. 1999. Propriétés et analyse d'un langage. In *TALN*.

Philippe Blache. 2001. *Les Grammaires de Propriétés : des contraintes pour le traitement automatique des langues naturelles*. Hermès Sciences.

Philippe Blache. 2004. Constraints: an operational framework for constructions grammars. In *ICCG-04*, pages 25–26.

Philippe Blache. 2005. Property Grammars: A fully constraint-based theory. In Henning Christiansen, Peter Rossen Skadhauge, and Jorgen Villadsen, editors, *Constraint Solving and Language Processing*, volume 3438 of *LNAI*. Springer.

Veronica Dahl and Philippe Blache. 2004. Directly executable constraint based grammars. In *Journees Francophones de Programmation en Logique avec Contraintes*, pages 149–166, Angers, France.

Ralph Debusmann, Denys Duchier, and Geert-Jan M. Kruijff. 2004. Extensible Dependency Grammar: A New Methodology. In *Proceedings of the 7th International Conference on Computational Linguistics (COLING 2004)*.

Denys Duchier. 1999. Axiomatizing Dependency Parsing Using Set Constraints. In *Proceedings 6th Meeting on the Mathematics of Language*, Orlando, FL.

Denys Duchier. 2000. Configuration Of Labeled Trees Under Lexicalized Constraints And Principles. To appear in the Journal of Language and Computation, December.

Kilian Foth, Wolfgang Menzel, and Ingo Schrd̈er. 2004. Robust Parsing with Weighted Constraints. *Natural Language Engineerings*.

Thom Frühwirth. 1994. Theory and Practice of Constraint Handling Rules. *The Journal of Logic Programming*, 37((1-3)), October. Special Issue on Constraint Logic Programming.

Edward Gibson. 2000. The Dependency Locality Theory: A Distance-Based Theory of Linguistic Complexity. In Alec Marantz, Yasushi Miyashita, and Wayne ONeil, editors, *Image, Language, Brain*, pages 95–126. Cambridge, Mass., MIT Press.

Adele Goldberg. 1995. *Constructions: A Construction Grammar Approach to Argument Structure*. Chicago University Press.

Johannes Heinecke, Jürgen Kunze, Wolfgang Menzel, and Ingo Shröder. 1998. Eliminative Parsing with Graded Constraints. In *Proc. 7th CoLing conf., 36th Annual Meeting of the ACL*, volume Coling–ACL '98, pages pp. 526–530, Montreal, Canada.

Frank Keller. 2000. *Gradience in Grammar - Experimental and Computational Aspects of Degrees of Grammaticality*. Ph.D. thesis, University of Edinburgh.

Hiroshi Maruyama. 1990. Structural Disambiguation with Constraint Propagation. In *Proceedings 28th Annual Meeting of the ACL*, pages pp. 31–38, Pittburgh, PA.

Frank Morawietz and Philippe Blache. 2002. Parsing natural languages with chr. Under consideration for publication in Theory and Practice of Logic Programming.

Tristan VanRullen. 2005. *Vers une analyse syntaxique à granularité variable*. Ph.D. thesis, Université de Provence, Informatique.

Gérard Verfaillie and Thomas Schiex. 1994. Solution reuse in dynamic CSPs. In *AAAI '94: Proc. of the twelfth national conf. on AI (vol. 1)*, pages 307–312, Menlo Park, CA, USA. American Ass. for AI.