Property Grammar Parsing Seen as a Constraint Optimization Problem

Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier, and Willy Lesaint

LIFO - Université d'Orléans - Bât. 3IA - Rue Léonard de Vinci
F-45 067 Orléans cedex 2, France
firstname.lastname@univ-orleans.fr

Abstract. Blache [1] introduced Property Grammar as a formalism where linguistic information is represented in terms of non hierarchical constraints. This feature gives it an adequate expressive power to handle complex linguistic phenomena, such as long distance dependencies, and also agrammatical sentences [2].
Recently, Duchier *et al.* [3] proposed a model-theoretic semantics for property grammar. The present paper follows up on that work and explains how to turn such a formalization into a constraint optimization problem, solvable using constraint programming techniques. This naturally leads to an implementation of a fully constraint-based parser for property grammars.

1 Introduction

Formal grammars typically limit their scope to well-formed utterances. As noted by [4], formal grammars in the style of *generative-enumerative syntax*, as they focus on generating well-formed models, are intrinsically ill-suited for providing accounts of ill-formed utterances. Formal grammars in the style of *model-theoretic syntax*, on the contrary, as they focus on judging models according to the constraints that they satisfy, are naturally well-suited to accommodate *quasi-expressions*.

Blache [2] proposed *Property Grammars* (PG) as a constrained-based formalism for analyzing both grammatical and agrammatical utterances. Prost [5] developed an approach based on PG and capable not only of providing analyses for any utterances, but also of making accurate judgements of grammaticality about them. Duchier *et al.* [3] provided model-theoretical semantics for PG and a formal logical account of Prost's work. In this paper, we show how such a formalization can be converted into a Constraint Optimization Problem, thus yielding a constraint-based parser that finds optimal parses using classical constraint programming techniques, such as branch-and-bound.

The use of constraint-based techniques for parsing is not new in itself, one may cite the seminal work of Duchier [6] on Dependency Grammar parsing, or that of Debusmann *et al.* [7] on Tree Adjoining Grammar parsing, or more

recently that of Parmentier and Maier [8], who proposed constraint-based extensions to Range Concatenation Grammar parsing. Nonetheless, PG parsing was lacking such a constraint-based axiomatization.[1]

The paper is organized as follows. We first introduce property grammars (section 2). In section 3, we then summarize the model-theoretic semantics of PG, as defined by Duchier *et al.* [3]. This semantics is used to define PG parsing as a Constraint Optimization Problem. This definition will be based on two types of constraints : tree-shapedness constraints, introduced in section 4, and property-related constraints, introduced in section 5. We then report on the implementation of a constraint-based parser using the Gecode library in section 6. In section 7, we compare our work with existing parsing environments for PG. Finally, in section 8, we conclude with some experimental results and hints about future work.

2 Property grammars

Property grammar [2] is a grammatical formalism where the relations between constituents are expressed in terms of local constraints[2], called properties, which can be independently violated. This makes it possible to describe agrammatical utterances (that is, whose description would not respect the whole set of constraints), and also to associate a given description with a grammaticality score (ratio between satisfied and unsatisfied constraints).

These constraints rely on linguistic observations, such as linear precedence between constituents, coocurrency between constituents, exclusion between constituents, *etc.* As suggested by Duchier *et al.* [3], a property grammar can be usefully understood as exploding classical phrase structure rules into collections of fine-grained properties. Each property has the form $A : \psi$ meaning that, in a syntactic tree, for a node of category A, the constraint ψ applies to its children.

For each node of category A, we consider the following properties:

Obligation	$A : \triangle B$	at least one B child
Uniqueness	$A : B!$	at most one B child
Linearity	$A : B \prec C$	B children precede C children

[1] Note that a first experiment of constraint-based axiomatization of PG was done by Dahl and Blache [9], we give more information on this later in section 7.

[2] Several attemps at characterizing syntactic trees through a system of constraints were developed in the late nineties. Among these, one may cite D-Tree Substitution Grammar (DSG) [10], and Tree Description Grammar (TDG) [11]. The main difference between these formalisms and PG is that the latter has been designed to provide a way to handle agrammatical sentences. Furthermore, in DSG and TDG, constraints are expressed using dominance-based tree descriptions, while PG's constraints are applied to syntactic categories.

Requirement	$A : B \Rightarrow C$	if a B child, then also a C child
Exclusion	$A : B \nRightarrow C$	B and C children are mutually exclusive
Constituency	$A : S$	children must have categories in S

As an example, let us consider the context free rules $\texttt{NP} \rightarrow \texttt{D N}$ and $\texttt{NP} \rightarrow \texttt{N}$ describing the relation between a noun and a determiner. They are translated into the following 7 properties:

$$_{(1)}\texttt{NP} : \{\texttt{D}, \texttt{N}\},\ _{(2)}\texttt{NP} : \texttt{D}!,\ _{(3)}\texttt{NP} : \triangle\texttt{N},\ _{(4)}\texttt{NP} : \texttt{N}!,\ _{(5)}\texttt{NP} : \texttt{D} \prec \texttt{N},\ _{(6)}\texttt{D} : \{\},\ _{(7)}\texttt{N} : \{\}.$$

(1) indicates that noun phrases only contain nouns or determiners, (2) states that in a noun phrase, there is at most one determiner. (3) and (4) say that in a noun phrase, there is exactly one noun. (5) indicates that, in a noun phrase, a determiner precedes a noun. Finally (6) and (7) state that determiners and nouns are leaf nodes in a valid syntactic tree.

In this context, if we only consider syntactic trees whose root has category NP, there are only two trees satisfying all properties:

We note that these syntactic trees are not lexicalized. In case we want to describe lexicalized trees, we can add some more lexical properties, such as

$$\texttt{cat}(\textit{apple}) = \texttt{N}$$

which defines the word *apple* as being a noun.

About the complexity of PG parsing Deep parsing with PG has been shown to be theoretically exponential in the number of categories of the grammar and the size of the sentence to parse [12]. As we shall see in section 7, existing approaches to deep parsing with PG usually rely on heuristics to reduce complexity in practice. In our approach, we want to avoid such heuristics. We are interested in studying the logical consequences of representation choices made in PG, while developing a parsing architecture for PG fully relying on constraint-satisfaction.

3 Model-theoretic semantics of property grammar

In this section, we give a summary of the model-theoretic semantics of PG developed by Duchier *et al.* [3].

First, recall that PGs are interpreted over syntactic tree structures. Two types of models are considered, according to whether we want to enforce the satisfaction of all grammatical properties or not : *strong* models and *loose* models (the latter corresponding to the modelization of agrammatical utterances).

Strong models. A syntax tree τ is a *strong* model of a grammar $\mathcal{G}$ iff for every node of τ and every property of $\mathcal{G}$, if that property is *pertinent* at that node, then it is also *satisfied*. The evaluation of the pertinence of a property depends on the type of the property. We consider 3 types of properties :

Type 1 : those which apply to a given node, such as *obligation*. For these properties, the pertinence only depends on the category of that node,

Type 2 : those which apply to a given couple of nodes (mother-daughter), such as *requirement* and *constituency*. For these properties, the pertinence depends on the category of the mother node, and that of its daughter nodes,

Type 3 : those which apply to a given triple of nodes (mother, daughter1, daughter2), such as *linearity*, *exclusion* and *uniqueness*. For these properties, the pertinence depends on the category of the mother node, and those of its daughter nodes.

Hence, when a node n has more than 2 children, a given property of type 3 has to be considered for every triple of nodes $(n, \_, \_)$. We call the pair consisting of a property ψ and such a tuple (*i.e.*, singleton, couple or triple of nodes), an *instance of property*. For example, the property $\texttt{NP} : \texttt{D} \prec \texttt{N}$ yields for every node as many instances as there are pairs of children. Such an instance is pertinent iff the mother node is of category `NP` and its children of categories `D` and `N` respectively. In addition, it is satisfied if the first child precedes the second one.

Later in this paper, we will represent the instance of a property $A : \psi$ at a node n using the following notation (n_x refers to a daughter node of n) :

$A : \psi@\langle n \rangle$	if ψ is of **type 1**
$A : \psi@\langle n, n_1 \rangle$	if ψ is of **type 2**
$A : \psi@\langle n, n_1, n_2 \rangle$	if ψ is of **type 3**

To sum up, every property is instantiated in every possible way at every node. Furthermore, as mentioned above, in a strong model every property instantiation that is pertinent has to be satisfied.

Loose models. Unlike strong models, in a *loose* model of a grammar, for a given utterance, every instance of property which is pertinent does not have to be satisfied. More precisely, a loose model is a syntax tree of maximal *fitness*, where fitness is the ratio of satisfied instances among those which are pertinent.

For a more detailed definition of this model-theoretic semantics of PG, we refer the reader to [3] . In the next sections, we show how this formalization of PG can be converted into a Constraint Optimization Problem, thus yielding a constraint-based parser that finds optimal parses.

4 Representing tree models using a grid

Our approach needs to enumerate candidate tree models, and to retain only those of maximal *fitness*. Since we do not know *a priori* the number of nodes of

our models, we propose to use a grid as a substrate, and to enumerate the trees which can be laid out on this grid.

For an utterance of m words, we know that each tree model has m leaves (PG do not use ϵ nodes). Unfortunately, we do not know the maximum depth of each tree model. We may use some heuristics to automatically assign a value n to the tree depth[3]. We chose to parametrize the associated parsing problem with a maximum tree depth n. Fixing this parameter allows us to layout a model over a subset of the nodes of an $n \times m$ grid. To represent our tree model, we will use a matrix $\mathcal{W}$ such that w_{ij} (with $1 \leq i \leq n$, and $1 \leq j \leq m$) refers to the node located at position (i, j) on the grid (rows and columns are numbered starting from 1, coordinate (1,1) being in the bottom-left corner). As an illustration of such a layout, see Fig. 1. We present in this section the constraints used to build a tree model on an $n \times m$ grid.

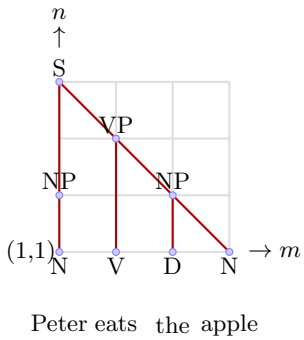


Fig. 1. Parse tree laid on a grid

Active nodes. Let V be the set of all nodes. A node is active if it is used by the model and inactive otherwise. We write V^+ for the set of active nodes and V^- for the rest. We have:

$$V = V^+ \uplus V^-$$

where $\uplus$ represents "disjoint union". Following the modeling technique of [13], for each node w, we write $\downarrow w$ for it's children, $\downarrow^+ w$ for its descendants, $\downarrow^* w$ for w and its descendants. Dually, we write $\uparrow w$ for w's parents, $\uparrow^+ w$ for its ancestors, $\uparrow^* w$ for w and its ancestors. Constraints relating these sets are:

$$\downarrow^+ w = \uplus\{\downarrow^* w' \mid w' \in \downarrow w\} \qquad \downarrow^* w = \{w\} \uplus \downarrow^+ w$$
$$\uparrow^+ w = \uplus\{\uparrow^* w' \mid w' \in \uparrow w\} \qquad \uparrow^* w = \{w\} \uplus \uparrow^+ w$$

[3] Due to the intrinsic recursive nature of language, the possibility to find an adequate depth value, *i.e.* not too big to prevent useless computations, and not too small to avoid missing solutions, is an open question.

Disjoint unions are justified by the fact that we are interested in tree models (*i.e.*, we do not allow for cycles). We additionally enforce the duality between ancestors and descendants:

$$w \in \uparrow w' \;\Leftrightarrow\; w' \in \downarrow w$$

and that each node has at most one parent:

$$|\uparrow w'| \leq 1$$

Inactive nodes have neither parents nor children:

$$w \in V^- \Rightarrow \downarrow w = \uparrow w = \emptyset$$

Since the root of the tree is still unknown, we write R for the set of root nodes. A tree model must have a single root:

$$|R| = 1$$

and the root node cannot be a child of any node:

$$V^+ = R \uplus (\uplus\{\downarrow w \mid w \in V\})$$

Projection. We write $\Downarrow w$ for the set of columns occupied by the tree anchored in w. As leaf nodes are located on the first row, their projection corresponds to their column, and only it :

$$\Downarrow w_{1j} = \{j\}$$

There are no interleaving projections (hence the disjoint union):

$$\Downarrow w_{ij} = \uplus\{\Downarrow w \mid w \in \downarrow w_{ij}\} \qquad 1 < j \leq m$$

There are no holes in the projection of any node (trees are projective):

$$\mathsf{convex}(\Downarrow w) \qquad \forall w \in V$$

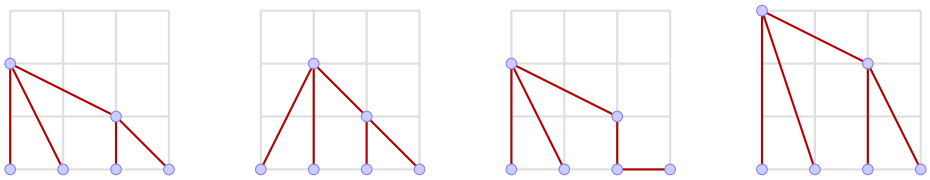
Dealing with symmetries. There are many ways of laying out a given tree on a grid. For instance, a four-node and three-leaf tree has among others the following layouts :

In order to have a unique way of laying out a tree, we add specific anti-symmetric constraints (the models satisfying these constraints are called *rectangular trees*) :

1. all leaves are located on the first row of the grid (*i.e.*, the bottom row),

2. the left-most daughter of any node is located on the same column as its mother node (this implies the subtree of a given node n occupies columns on the right of n),
3. every node is above any of its descendant nodes (this implies the subtree of a given node n occupies rows that are below that of n),
4. every internal node must have a daughter node in the row directly below (this implies there are no empty rows below the root node).

As an illustration, among the following trees, only the first one is a rectangular tree (the second tree violates condition 2, the third one condition 3 and the fourth one condition 4) :



How these 4 conditions are represented in our axiomatization ? First, let us write $c(w)$ for the column of node w and $\ell(w)$ for its line:

$$c(w_{ij}) = j \qquad\qquad \ell(w_{ij}) = i$$

(1) The words are linked to the bottom row of the grid, which contains the leaves of the tree. These must all be active:

$$\{w_{1j} \mid 1 \leq j \leq m\} \subseteq V^+$$

(2) Any active node must be placed in the column of the left-most leaf of its subtree:

$$w_{ij} \in V^+ \quad \Leftrightarrow \quad j = \min \Downarrow w_{ij}$$

This stipulation and the fact (3) every node is above of its descendants are translated by constraints on the domains of variables. As mentioned above, the descendants of a node n are on the down-right part of the grid with respect to n. The dual holds, that is the ancestors of a node n are on the upper-left part of the grid with respect to n:

$$\downarrow^+ w_{ij} \subseteq \{w_{lk} \mid 1 \leq l < i,\ j \leq k \leq m\}$$
$$\uparrow^+ w_{ij} \subseteq \{w_{lk} \mid i < l \leq n,\ 1 \leq k \leq j\}$$

(4) Any active non-bottom node has at least one child at the level just below:

$$w_{ij} \in V^+ \quad \Leftrightarrow \quad i - 1 \in \{\ell(w) \mid w \in \downarrow w_{ij}\} \qquad\qquad 1 < i \leq n$$

Categories. In order to model syntax trees, we also need to assign to each active node a syntactic category. For simplicity, we will assign the category `none` to all and only the inactive nodes:

$$\mathsf{cat}(w) = \texttt{none} \quad \Leftrightarrow \quad w \in V^-$$

For active nodes, the category will be assigned via property-related constraints, which are introduced in the next section. Finally, words are related to leaves via their category:

$$\mathsf{cat}(w_{1j}) = \mathsf{cat}(\text{word}_j)$$

where word_j refers to the j^{th} word of the sentence to parse.

5 Handling instances of properties

Recall that each property has the form $A : \psi$, which means that for a node of category A, the constraint ψ applies to its children. For example the property $A : B \prec C$ is intended to mean that, for a non-leaf node of category A and any two daughters of this node labeled respectively with categories B and C, then the one labelled with B must precede the one labeled with C. Clearly, for each node of category A, this property must be checked for every pair of its daughters. This corresponds to the notion of instances of a property introduced earlier in section 3.

An instance of a property is a pair of the property and a tuple of nodes to which it is applied. An instance is pertinent if the node where it is instantiated is active (*i.e.*, belongs to V^+) and the parameter nodes of its tuple have the categories stipulated in the property. An instance is satisfied if the property is satisfied. For each instance I we define two boolean variables $P(I)$ and $S(I)$ denoting respectively its *pertinence* and its *pertinence and satisfaction*.

In the following paragraphs,we translate each property of PG into a set of constraints for our constraint optimization problem.

Properties of type 1 Let us start with properties of type 1, that is to say, whose instance depends on a single node. The only such property is obligation.

Obligation. The property $A : \triangle B$ yields instances I of the form:

$$(A : \triangle B)@\langle w_{i_0 j_0} \rangle$$

It is pertinent if $w_{i_0 j_0}$ is an active node labelled with A:

$$P(I) \quad \Leftrightarrow \quad (\, w_{i_0 j_0} \in V^+ \wedge cat(w_{i_0 j_0}) = A \,)$$

It is satisfied if at least one of its children is labelled with B:

$$S(I) \quad \Leftrightarrow \quad (\, P(I) \wedge \bigvee_{w_{ij} \in \downarrow w_{i_0 j_0}} cat(w_{ij}) = B \,)$$

Properties of type 2 Let us continue with properties of type 2, whose instance depends on a couple of nodes. These corresponds to requirement and constituency.

Requirement. The property $A : B \Rightarrow C$ yields instances I of the form:

$$(A : B \Rightarrow C)@\langle w_{i_0j_0}, w_{i_1j_1}\rangle$$

It is pertinent only if $w_{i_0j_0}$ is active and $w_{i_1j_1}$ is one of its children and their categories correspond:

$$P(I) \Leftrightarrow \begin{pmatrix} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in {\downarrow}w_{i_0j_0} \wedge \\ \mathsf{cat}(w_{i_0j_0}) = A \wedge \mathsf{cat}(w_{i_1j_1}) = B \end{pmatrix}$$

It is satisfied if one of $w_{i_0j_0}$'s children is labelled with C:

$$S(I) \quad \Leftrightarrow \quad (\, P(I) \wedge \bigvee_{w_{ij} \in {\downarrow}w_{i_0j_0}} \mathsf{cat}(w_{ij}) = C \,)$$

Constituency. The property $A : S$ yields instances I of the form:

$$(A : B \prec C)@\langle w_{i_0j_0}, w_{i_1j_1}\rangle$$

It is pertinent only if $w_{i_0j_0}$ is active and labelled with A and $w_{i_1j_1}$ is one of its children:

$$P(I) \Leftrightarrow \begin{pmatrix} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in {\downarrow}w_{i_0j_0} \wedge \\ \mathsf{cat}(w_{i_0j_0}) = A \end{pmatrix}$$

It is satisfied if the category of $w_{i_1j_1}$ is in S:

$$S(I) \quad \Leftrightarrow \quad (\, P(I) \wedge \mathsf{cat}(w_{i_1j_1}) \in S \,)$$

Properties of type 3 Let us finish with properties of type 3, whose instance depends on a triple of nodes. These properties are linearity, uniqueness and exclusion.

Linearity. The property $A : B \prec C$ yields instances I of the form:

$$(A : B \prec C)@\langle w_{i_0j_0}, w_{i_1j_1}, w_{i_2j_2}\rangle$$

I is pertinent if $w_{i_0j_0}$ is active, $w_{i_1j_1}$ and $w_{i_2j_2}$ are its children, and each node is labelled with the corresponding category:

$$P(I) \Leftrightarrow \begin{pmatrix} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in {\downarrow}w_{i_0j_0} \wedge \\ w_{i_2j_2} \in {\downarrow}w_{i_0j_0} \wedge \mathsf{cat}(w_{i_0j_0}) = A \wedge \\ \mathsf{cat}(w_{i_1j_1}) = B \wedge \mathsf{cat}(w_{i_2j_2}) = C \end{pmatrix}$$

Its satisfaction depends on whether the node $w_{i_1j_1}$ precedes $w_{i_2j_2}$ or not. It is thus defined as:

$$S(I) \quad \Leftrightarrow \quad (\, P(I) \wedge j_1 < j_2 \,)$$

Uniqueness. The property $A : B!$ yields instances I of the form:

$$(A : B!)@\langle w_{i_0j_0}, w_{i_1j_1}, w_{i_2j_2}\rangle$$

It is active only if $w_{i_0j_0}$ is active and labelled with A and $w_{i_1j_1}$ and $w_{i_2j_2}$ are its children and are labelled with B:

$$P(I) \Leftrightarrow \begin{pmatrix} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in {\downarrow}w_{i_0j_0} \wedge \\ w_{i_2j_2} \in {\downarrow}w_{i_0j_0} \wedge \mathsf{cat}(w_{i_0j_0}) = A \wedge \\ \mathsf{cat}(w_{i_1j_1}) = B \wedge \mathsf{cat}(w_{i_2j_2}) = B \end{pmatrix}$$

It is satisfied if $w_{i_1j_1}$ and $w_{i_2j_2}$ are the same node:

$$S(I) \quad \Leftrightarrow \quad (\, P(I) \wedge w_{i_1j_1} = w_{i_2j_2} \,)$$

Exclusion. The property $A : B \not\Leftrightarrow C$ yields instances I of the form:

$$(A : B \not\Leftrightarrow C)@\langle w_{i_0j_0}, w_{i_1j_1}, w_{i_2j_2}\rangle$$

It is pertinent only if $w_{i_0j_0}$ is active and labelled with A and $w_{i_1j_1}$ and $w_{i_2j_2}$ are its children where either $w_{i_1j_1}$ is labelled with B or $w_{i_2j_2}$ is labelled with C:

$$P(I) \Leftrightarrow \begin{pmatrix} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in {\downarrow}w_{i_0j_0} \wedge \\ w_{i_2j_2} \in {\downarrow}w_{i_0j_0} \wedge \mathsf{cat}(w_{i_0j_0}) = A \wedge \\ (\mathsf{cat}(w_{i_1j_1}) = B \vee \mathsf{cat}(w_{i_2j_2}) = C) \end{pmatrix}$$

Its satisfaction relies on the fact that the two children does not both have the incompatible categories:

$$S(I) \Leftrightarrow P(I) \wedge (\, \mathsf{cat}(w_{i_1j_1}) \neq B \vee \mathsf{cat}(w_{i_2j_2}) \neq C \,)$$

In other terms, if $w_{i_1j_1}$ is labelled with B then $w_{i_2j_2}$ cannot be labelled with C, and if $w_{i_2j_2}$ is labelled with C then $w_{i_1j_1}$ cannot be labelled with B.

As was mentioned in section 3, in the loose semantics of PG, we want to compute models with the best fitness. To do this, we add an optimization constraint.

Optimization constraint. To account for the loose semantics of PG, a property instance counts if it is pertinent, it counts positively if satisfied, negatively otherwise. Let $\mathcal{I}$ be the set of all property instances, $\mathcal{I}^0$ the subset of pertinent instances and $\mathcal{I}^+$ the subset of positive instances. We want to find models which maximize the ratio $|\mathcal{I}^+|/|\mathcal{I}^0|$.

Since for each instance I, the variables $P(I)$ and $S(I)$ are boolean, their reified value is either 0 or 1. We can calculate the cardinality of these sets the following way:

$$|\mathcal{I}^0| = \sum_{I \in \mathcal{I}} P(I) \qquad\qquad |\mathcal{I}^+| = \sum_{I \in \mathcal{I}} S(I)$$

6 Implementation

The approach described so far has been implemented using the Gecode constraint programming library [14].

Each node w_{ij} of the grid is identified with an integer $k = (i-1) \times m + j$ (where m is the width of the grid). The set of nodes V is defined as $V = \{1, \ldots, n \times m\}$. V^+ and V^- are two set variables such that $V^+, V^- \subseteq V$. All the constraints related to these sets are implemented using Gecode's API. The relations $\downarrow, \downarrow^+, \downarrow^*$ and $\uparrow, \uparrow^+, \uparrow^*$ are encoded using arrays of set variables, whose indexes are nodes of the grid. We also use arrays of set variables to encode property-related constraints. As there are many types of constraints and many instances to consider, the computation of the indexes is slightly more complex than the ones used for tree-shapedness constraints. Definitions of $P(I)$ and $S(I)$ are realized using *reified constraints*. The search for an optimal parse is achieved using the *branch-and-bound* search strategy to maximize the ratio $|\mathcal{I}^+|/|\mathcal{I}^0|$.

An example search tree for the grammatical utterance "`Peter eats the apple`" is given in Fig. 2. The graphical representation of the search tree has been built by Gist, the Gecode Interactive Search Tool [15]. On this figure, round nodes represent choice points, square and triangle-shaped nodes failures, and diamond-shaped nodes solutions of the constraint optimization problem (in this example, the last diamond on the right refers to the optimal solution).

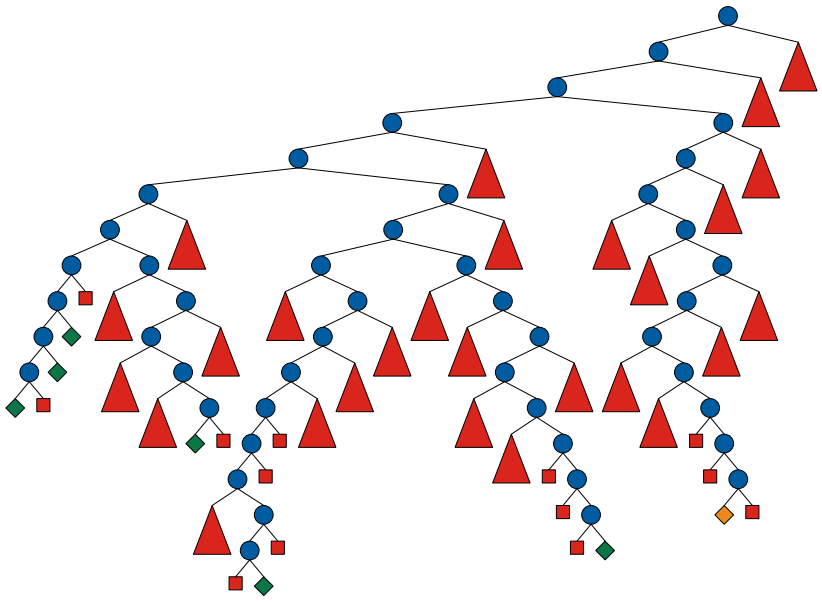


Fig. 2. Example search tree

To give an illustration of the complexity of the constraint optimization problem[4] for the parse example of Fig. 2 (sentence "`Peter eats the apple`", and grammar having 19 properties handling 6 categories), the search tree has about 450,000 nodes, among which 7 are solutions. The optimal syntactic tree is represented in Fig. 3.

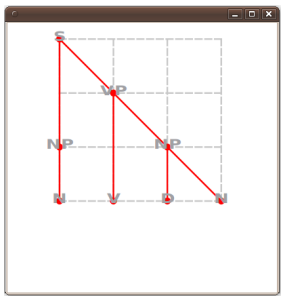


Fig. 3. Optimal syntactic tree for "Peter eats the apple"

As the parser is still in an early development stage, we do not have any benchmark. As mentioned above, there are many instances of property to handle, that is to say, many constraints to evaluate. In practice, our parser can relatively quickly find a syntactic tree (in less than a second for the example above), but the proof of optimality can take about a minute.[5] While we are mainly interested in exploring the logical consequences of representation choices made in PG (that is, without using any heuristic to reduce complexity), we are also interested in improving the computation of optimal parses, either by:

- parallelizing the exploration of the search tree,
- or enriching the information associated with lexical entries, for example by using a Part-Of-Speech tagger.

The parser is freely available on demand, and released under the terms of the GNU General Public License.

7 Comparison with existing work

Among the different approaches to PG parsing, one may cite the seminal work of Blache and Balfourier [16]. This work was later followed by a series of papers by Dahl and Blache [9], Estratat and Henocque [17], Van Rullen [12, 18], Blache and Rauzy [19], and more recently Prost [5].

[4] Our PG parsing algorithm using constraint-satisfaction is clearly exponential as all candidate trees are enumerated.

[5] These results are obtained on a 2.6 Ghz processor with 4 Gb of RAM.

The main difference between these approaches and our work, is that, apart from [9] and [17], they do not rely on a model-theoretic formal semantics of PG. They rather apply well-known efficient parsing techniques and heuristics to PG. Thus, [16] uses a constraint selection process to incrementally build the syntactic tree of a sentence. [12, 18] include hybrid approaches mixing deep and shallow parsing. In [19], the authors propose to extend symbolic parsing with probabilities on syntactic categories. [5] uses a chart-based parsing algorithm, where the items contain optimal sub-trees, used to derive a complete syntactic tree.

A first attempt to use a constraint satisfaction-based approach to PG parsing is [9]. In their work, the authors encode the input PG into a set of rules for the *Constraint Handling Rule* system [20]. Their encoding makes it possible to directly interpret the PG in terms of satisfied / relaxed constraints on syntactic categories. On top of this interpretation, they use rewriting rules to propagate constraint satisfaction / relaxation, and a syntactic tree is built as a side effect. The main difference with our approach lies in the fact that the authors control the way a constraint is selected for evaluation, while we rely on classical constraint-based techniques such as branch-and-bound to select and propagate constraint evaluations. That is, we clearly distinguish the definition of the constraint satisfaction problem from its resolution.

Another constraint-based approach to PG parsing is [17]. In their work, the authors translate a PG into a model in the Object Constraint Language (OCL). This model is interpreted as a configuration problem, which is fed to a configurator. The latter solves the constraints lying in the input model. The result is a valid syntactic structure. Contrary to our approach, or that of [9], this OCL-encoding does not allow for relaxed constraints. Hence, it only computes syntactic structures that satisfy the whole set of constraints. In other terms, it cannot make full advantage of the PG formalism, which describes natural language in terms of local constraints that can be violated. This feature is particularly useful when dealing with agrammatical sentences such as those spoken language often contain.

8 Conclusion

Duchier *et al.* [3] provided precise model-theoretical semantics for property grammars. In this paper, we extend that work and show how such a formalization can be converted into a Constraint Optimization Problem, thus yielding a constraint-based parser capable of finding optimal parses using classical constraint-based techniques such as branch-and-bound. Furthermore, we have implemented this convertion and are able to experiment with analyzing both grammatical and agrammatical utterances.

The work described here is still at an early stage of development. It is not intended to compete with high-performance parsers, but rather to serve as an experimental platform for grammar development and linguistic modeling, where

logical consequences are not accidentally hidden by the effect of performance-oriented heuristics.

In a near future, we plan to work on the definition and implementation of an extension to branch-and-bound, in order to keep not only one but all syntactic trees having the maximum fitness.

Acknowledgements

We are grateful to Sylvie Billot, Matthieu Lopez, Jean-Philippe Prost, Isabelle Tellier and three anonymous reviewers for useful comments on this work.

References

1. Blache, P.: Constraints, Linguistic Theories and Natural Language Processing. Lecture Notes in Artificial Intelligence Vol. 1835. Springer-Verlag (2000)
2. Blache, P.: Property Grammars: a Fully Constraint-Based Theory. In H. Christiansen, P. Rossen Skadhauge, J.V., ed.: Constraint Solving and Language Processing. Springer (2004) 1–16 Lecture Notes in Artificial Intelligence, Vol. 3438.
3. Duchier, D., Prost, J.P., Dao, T.B.H.: A model-theoretic framework for grammaticality judgements. In: Conference on Formal Grammar (FG2009), Bordeaux, France (July 2009)
4. Pullum, G., Scholz, B.: On the Distinction Between Model-Theoretic and Generative-Enumerative Syntactic Frameworks. In de Groote, P., Morrill, G., Rétoré, C., eds.: Logical Aspects of Computational Linguistics: 4th International Conference. Number 2099 in Lecture Notes in Artificial Intelligence, Berlin, Springer Verlag (2001) 17–43
5. Prost, J.P.: Modelling Syntactic Gradience with Loose Constraint-based Parsing. Cotutelle Ph.D. Thesis, Macquarie University, Sydney, Australia, and Université de Provence, Aix-en-Provence, France (December 2008)
6. Duchier, D.: Axiomatizing Dependency Parsing Using Set Constraints. In: Sixth Meeting on Mathematics of Language, Orlando, Florida (July 1999) 115–126
7. Debusmann, R., Duchier, D., Kuhlmann, M., Thater, S.: TAG Parsing as Model Enumeration. In: Proceedings of the 7th international Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+7), Vancouver, Canada (2004) 148–154
8. Parmentier, Y., Maier, W.: Using Constraints over Finite Sets of Integers for Range Concatenation Grammar Parsing. In: Proceedings of the 6th International Conference on Natural Language Processing, GoTAL 2008, Advances in Natural Language Processing, Springer Berlin-Heidelberg (2008) 360–365
9. Dahl, V., Blache, P.: Directly Executable Constraint Based Grammars. In: Actes des Journées Francophones de Programmation Logique et par Contraintes 2004 (JFPLC-04), Angers, France (2004)
10. Rambow, O., Weir, D., Vijay-Shanker, K.: D-tree substitution grammars. Computational Linguistics **27**(1) (2001) 89–121
11. Kallmeyer, L.: Local Tree Description Grammars: A local extension of TAG allowing underspecified dominance relations. Grammars **4** (2001) 85–137
12. van Rullen, T.: Vers une analyse syntaxique à granularité variable. PhD thesis, Université de Provence, Aix-Marseille 1, France (2005)

13. Duchier, D.: Configuration Of Labeled Trees Under Lexicalized Constraints And Principles. Journal of Research on Language and Computation **1**(3/4) (September 2003) 307–336
14. Gecode Team: Gecode: Generic constraint development environment (2010) Available from `http://www.gecode.org`.
15. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and Programming with Gecode (2010) Available from `http://www.gecode.org/doc-latest/MPG.pdf`.
16. Blache, P., Balfourier, J.M.: Property Grammars: a Flexible Constraint-Based Approach to Parsing. In: International Workshop on Parsing Techniques, Beijing, China (2001)
17. Estratat, M., Henocque, L.: Parsing languages with a configurator. In: Proceedings of the European Conference for Artificial Intelligence ECAI'2004, Valencia, Spain (August 2004) 591–595
18. Vanrullen, T., Blache, P., Balfourier, J.M.: Constraint-Based Parsing as an Efficient Solution: Results from the Parsing Evaluation Campaign EASy. In: Proceedings of the Language and Resources Evaluation Conference, Genoa, Italy (2006)
19. Blache, P., Rauzy, S.: Mécanismes de contrôle pour l'analyse en Grammaires de Propriétés. In: Actes, Traitement Automatique des Langues Naturelles (TALN), P. Mertens, C. Fairon, A. Dister et P. Watrin eds. (2006) 415–424
20. Frühwirth, T.: Constraint Handling Rules. Cambridge University Press (2009) ISBN 9780521877763.