

# Generating with a Grammar Based on Tree Descriptions: a Constraint-Based Approach

Claire Gardent

CNRS

LORIA, BP 239 Campus Scientifique  
54506 Vandoeuvre-les-Nancy, France  
claire.gardent@loria.fr

Stefan Thater

Computational Linguistics

Universität des Saarlandes

Saarbrücken, Germany

stth@coli.uni-sb.de

## Abstract

While the *generative* view of language processing builds bigger units out of smaller ones by means of rewriting steps, the *axiomatic* view eliminates invalid linguistic structures out of a set of possible structures by means of well-formedness principles. We present a generator based on the axiomatic view and argue that when combined with a TAG-like grammar and a flat semantics, this axiomatic view permits avoiding drawbacks known to hold either of top-down or of bottom-up generators.

## 1 Introduction

We take the axiomatic view of language and show that it yields an interestingly new perspective on the tactical generation task i.e. the task of producing from a given semantics  $\phi$  a string with semantics  $\phi$ .

As (Cornell and Rogers, To appear) clearly shows, there has recently been a surge of interest in logic based grammars for natural language. In this branch of research sometimes referred to as “Model Theoretic Syntax”, a grammar is viewed as a set of axioms defining the well-formed structures of natural language.

The motivation for model theoretic grammars is initially theoretical: the use of logic should support both a more precise formulation of grammars and a different perspective on the mathematical and computational properties of natural language.

But eventually the question must also be addressed of how such grammars could be put to work. One obvious answer is to use a model generator. Given a logical formula  $\phi$ , a model genera-

tor is a program which builds some of the models satisfying this formula. Thus for parsing, a model generator can be used to enumerate the (minimal) model(s), that is, the parse trees, satisfying the conjunction of the lexical categories selected on the basis of the input string plus any additional constraints which might be encoded in the grammar. And similarly for generation, a model generator can be used to enumerate the models satisfying the bag of lexical items selected by the lexical look up phase on the basis of the input semantics.

How can we design model generators which work efficiently on natural language input i.e. on the type of information delivered by logic based grammars? (Duchier and Gardent, 1999) shows that constraint programming can be used to implement a model generator for tree logic (Backofen et al., 1995). Further, (Duchier and Thater, 1999) shows that this model generator can be used to *parse* with descriptions based grammars (Rambow et al., 1995; Kallmeyer, 1999) that is, on logic based grammars where lexical entries are descriptions of trees expressed in some tree logic.

In this paper, we build on (Duchier and Thater, 1999) and show that modulo some minor modifications, the same model generator can be used to *generate* with description based grammars. We describe the workings of the algorithm and compare it with standard existing top-down and bottom-up generation algorithms. In specific, we argue that the change of perspective offered by the constraint-based, axiomatic approach to processing presents some interesting differences with the more traditional *generative* approach usually pursued in tactical generation and further, that the combination of this static view with a TAG-like grammar and a flat semantics results in a system which combines the positive aspects of both top-

down and bottom-up generators.

The paper is structured as follows. Section 2 presents the grammars we are working with namely, Description Grammars (DG), Section 3 summarises the parsing model presented in (Duchier and Thater, 1999) and Section 4 shows that this model can be extended to generate with DGs. In Section 5, we compare our generator with top-down and bottom-up generators, Section 6 reports on a proof-of-concept implementation and Section 7 concludes with pointers for further research.

## 2 Description Grammars

There is a range of grammar formalisms which depart from Tree Adjoining Grammar (TAG) by taking as basic building blocks tree descriptions rather than trees. D-Tree Grammar (DTG) is proposed in (Rambow et al., 1995) to remedy some empirical and theoretical shortcomings of TAG; Tree Description Grammar (TDG) is introduced in (Kallmeyer, 1999) to support syntactic and semantic underspecification and Interaction Grammar is presented in (Perrier, 2000) as an alternative way of formulating linear logic grammars.

Like all these frameworks, DG uses tree descriptions and thereby benefits first, from the extended domain of locality which makes TAG particularly suitable for generation (cf. (Joshi, 1987)) and second, from the monotonicity which differentiates descriptions from trees with respect to adjunction (cf. (Vijay-Shanker, 1992)).

DG differs from DTG and TDG however in that it adopts an axiomatic rather than a generative view of grammar: whereas in DTG and TDG, derived trees are constructed through a sequence of rewriting steps, in DG derived trees are models satisfying a conjunction of elementary tree descriptions. Moreover, DG differs from Interaction Grammars in that it uses a flat rather than a Montague style recursive semantics thereby permitting a simple syntax/semantics interface (see below).

A Description Grammar is a set of lexical entries of the form  $\langle \tau, \sigma \rangle$  where  $\tau$  is a tree description and  $\sigma$  is the semantic representation associated with  $\tau$ .

**Tree descriptions.** A tree description is a conjunction of literals that specify either the label of a node or the position of a node relative to

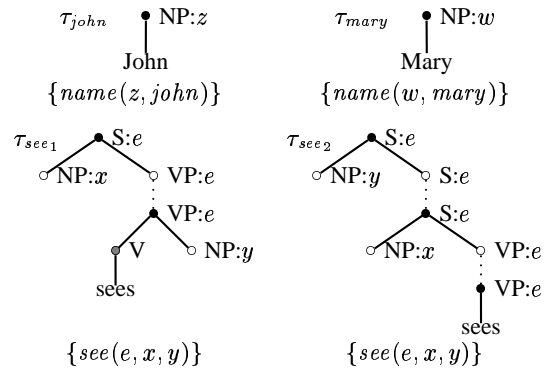


Figure 1: Example grammar 1

other nodes. As a logical notation quickly becomes unwieldy, we use graphics instead. Figure 1 gives a graphic representation of a small DG fragment. The following conventions are used. Nodes represent node variables, plain edges strict dominance and dotted edges dominance. The labels of the nodes abbreviate a feature structure, e.g. the label  $NP:x$  represents the feature structure  $\{cat:np, idx:x\}$ , while the anchor represents the *phon* value in the feature structure of the immediately dominating node variable.

Node variables can have positive, negative or neutral polarity which are represented by black, white and gray nodes respectively. Intuitively, a negative node variable can be thought of as an open valency which must be filled exactly once by a positive node variable while a neutral node variable is a variable that may not be identified with any other node variable. Formally, polarities are used to define the class of saturated models. A *saturated model*  $M$  for a tree description  $\tau$  (written  $M \models_S \tau$ ) is a model in which each negative node variable is identified with exactly one positive node variable, each positive node variable with exactly one negative node variable and neutral node variables are not identified with any other node variable. Intuitively, a saturated model for a given tree description is the smallest tree satisfying this description and such that all syntactic valencies are filled. In contrast, a *free model*  $M$  for  $\tau$  (written,  $M \models_F \tau$ ) is a model such that every node in that model interprets exactly one node variable in  $\tau$ .

In DG, lexical tree descriptions must obey the following conventions. First, the polarities are used in a systematic way as follows. Roots of

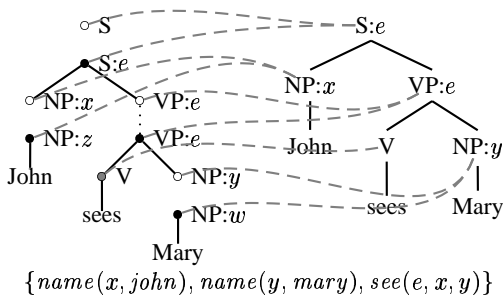


Figure 2:  $M \models_S \tau_{see_1} \wedge \tau_{john} \wedge \tau_{mary}$

fragments (fully specified subtrees) are always positive; except for the anchor, all leaves of fragments are negative, and internal node variables are neutral. This guarantees that in a saturated model, tree fragments that belong to the denotation of distinct tree descriptions do not overlap. Second, we require that every lexical tree description has a single minimal free model, which essentially means that the lexical descriptions must be tree shaped.

**Semantic representation.** Following (Stone and Doran, 1997), we represent meaning using a flat semantic representation, i.e. as multisets, or conjunctions, of non-recursive propositions. This treatment offers a simple syntax-semantics interface in that the meaning of a tree is just the conjunction of meanings of the lexical tree descriptions used to derive it once the free variables occurring in the propositions are instantiated. A free variable is instantiated as follows: each free variable labels a syntactic node variable  $n$  and is unified with the label of any node variable identified with  $n$ . For the purpose of this paper, a simple semantic representation language is adopted which in particular, does not include “handles” i.e. labels on propositions. For a wider empirical coverage including e.g. quantifiers, a more sophisticated version of flat semantics can be used such as Minimal Recursion Semantics (Copestake et al., 1999).

### 3 Parsing with DG

Parsing with DG can be formulated as a model generation problem, the task of finding models satisfying a give logical formula. If we restrict our attention to grammars where every lexical tree description has exactly one anchor and (unrealistically) assuming that each word is associated

—	$\tau_{root}$	<i>true</i>
John	$\tau_{john}$	<i>true</i>
saw	$\tau_{see_1}$	$\tau_{see_2}$
Mary	$\tau_{mary}$	<i>true</i>

Figure 3: Example parsing matrix

with exactly one lexical entry, then parsing a sentence  $w_1 \dots w_n$  consists in finding the saturated model(s)  $M$  with yield  $w_1 \dots w_n$  such that  $M$  satisfies the conjunction of lexical tree descriptions  $\phi_1 \wedge \dots \wedge \phi_n$  with  $\phi_i$  the tree description associated with the word  $w_i$  by the grammar.

Figure 2 illustrates this idea for the sentence “John loves Mary”. The tree on the right hand side represents the saturated model satisfying the conjunction of the descriptions given on the left and obtained from parsing the sentence “John sees Mary” (the isolated negative node variable, the “ROOT description”, is postulated during parsing to cancel out the negative polarity of the top-most S-node in the parse tree). The dashed lines between the left and the right part of the figure schematise the interpretation function: it indicates which node variables gets mapped to which node in the model.

As (Duchier and Thater, 1999) shows however, lexical ambiguity means that the parsing problem is in fact more complex as it in effect requires that models be searched for that satisfy a conjunction of disjunctions (rather than simply a conjunction) of lexical tree descriptions.

The constraint based encoding of this problem presented in (Duchier and Thater, 1999) can be sketched as follows<sup>1</sup>. To start with, the conjunction of disjunctions of descriptions obtained on the basis of the lexical lookup is represented as a matrix, where each row corresponds to a word from the input (except for the first row which is filled with the above mentioned ROOT description) and columns give the lexical entries associated by the grammar with these words. Any matrix entry which is empty is filled with the formula *true* which is true in all models. Figure 3 shows an example parsing matrix for the string “John saw Mary” given the grammar in Figure 1.<sup>2</sup>

Given such a matrix, the task of parsing con-

<sup>1</sup>For a detailed presentation of this constraint based encoding, see the paper itself.

<sup>2</sup>For lack of space in the remainder of the paper, we omit the ROOT description in the matrices.

sists in:

1. selecting exactly one entry per row thereby producing a conjunction of selected lexical entries,
2. building a saturated model for this conjunction of selected entries such that the yield of that model is equal to the input string and
3. building a free model for each of the remaining (non selected) entries.

The important point about this way of formulating the problem is that it requires all constraints imposed by the lexical tree descriptions occurring in the matrix to be satisfied (though not necessarily in the same model). This ensures strong constraint propagation and thereby reduces non-determinism. In particular, it avoids the combinatorial explosion that would result from first generating the possible conjunctions of lexical descriptions out of the CNF obtained by lexical lookup and second, testing their satisfiability.

## 4 Generating with DG

We now show how the parsing model just described can be adapted to generate from some semantic representation  $\phi$ , one or more sentence(s) with semantics  $\phi$ .

### 4.1 Basic Idea

The parsing model outlined in the previous section can directly be adapted for generation as follows. First, the lexical lookup is modified such that propositions instead of words are used to determine the relevant lexical tree descriptions: a lexical tree description is selected if its semantics subsumes part of the input semantics. Second, the constraint that the yield of the saturated model matches the input string is replaced by a constraint that the sum of the cardinalities of the multisets of propositions associated with the lexical tree descriptions composing the solution tree equals the cardinality of the input semantics. Together with the above requirement that only lexical entries be selected whose semantics subsumes part of the goal semantics, this ensures that the semantics of the solution trees is identical with the input semantics.

The following simple example illustrates this idea. Suppose the input semantics is

$\{name(x, john), name(y, mary), see(e, x, y)\}$  and the grammar is as given in Figure 1. The generating matrix then is:

$name(x, john)$	$\tau_{john}$	$true$
$see(e, x, y)$	$\tau_{see_1}$	$\tau_{see_2}$
$name(y, mary)$	$\tau_{mary}$	$true$

Given this generating matrix, two matrix models will be generated, one with a saturated model  $M_3$  satisfying  $\tau_{john} \wedge \tau_{see_1} \wedge \tau_{mary}$  and a free model satisfying  $\tau_{see_2}$  and the other with the saturated model  $M_4$  satisfying  $\tau_{john} \wedge \tau_{see_2} \wedge \tau_{mary}$  and a free model satisfying  $\tau_{see_1}$ . The first solution yields the sentence “John sees Mary” whereas the second yields the topicalised sentence “Mary, John sees.”

### 4.2 Going Further

The problem with the simple method outlined above is that it severely restricts the class of grammars that can be used by the generator. Recall that in (Duchier and Thater, 1999)’s parsing model, the assumption is made that each lexical entry has exactly one anchor. In practice this means that the parser can deal neither with a grammar assigning trees with multiple anchors to idioms (as is argued for in e.g. (Abeillé and Schabes, 1989)) nor with a grammar allowing for trace anchored lexical entries. The mirror restriction for generation is that each lexical entry must be associated with exactly one semantic proposition. The resulting shortcomings are that the generator can deal neither with a lexical entry having an empty semantics nor with a lexical entry having a multi-propositional semantics. We first show that these restrictions are too strong. We then show how to adapt the generator so as to lift them.

**Empty Semantics.** Arguably there are words such as “that” or infinitival “to” whose semantic contribution is void. As (Shieber, 1988) showed, the problem with such words is that they cannot be selected on the basis of the input semantics. To circumvent this problem, we take advantage of the TAG extended domain of locality to avoid having such entries in the grammar. For instance, complementizer “that” does not anchor a tree description by itself but occurs in all lexical tree descriptions providing an appropriate syntactic context for it, e.g. in the tree description for “say”.

**Multiple Propositions.** Lexical entries with a multi-propositional semantics are also very common. For instance, a neo-Davidsonian semantics would associate e.g.  $run(e)$ ,  $agent(e, x)$  with the verb “run” or  $run(e, x)$ ,  $past(e)$  with the past tensed “ran”. Similarly, agentless passive “be” might be represented by an overt quantification over the missing agent position (such as  $\exists e \exists x. R(e) \wedge agent(e, x)$  with  $R$  a variable over the complement verb semantics). And a grammar with a rich lexical semantics might for instance associate the semantics  $want(e_1, x, e_2)$ ,  $have(e_2, x)$  with “want” (cf. (McCawley, 1979) which argues for such a semantics to account for examples such as “Reuters wants the report tomorrow” where “tomorrow” modifies the “having” not the “wanting”).

Because it assumes that each lexical entry is associated with exactly one semantic proposition, such cases cannot be dealt with the generator sketched in the previous section. A simple method for fixing this problem would be to first partition the input semantics in as many ways as are possible and to then use the resulting partitions as the basis for lexical lookup.

The problems with this method are both theoretical and computational. On the theoretical side, the problem is that the partitioning is made independent of grammatical knowledge. It would be better for the decomposition of the input semantics to be specified by the lexical lookup phase, rather than by means of a language independent partitioning procedure. Computationally, this method is unsatisfactory in that it implements a generate-and-test procedure (first, a partition is created and second, model generation is applied to the resulting matrices) which could rapidly lead to combinatorial explosion and is contrary in spirit to (Duchier and Thater, 1999) constraint-based approach.

We therefore propose the following alternative procedure. We start by marking in each lexical entry, one proposition in the associated semantics as being the *head* of this semantic representation. The marking is arbitrary: it does not matter which proposition is the head as long as each semantic representation has exactly one head. We then use this head for lexical lookup. Instead of selecting lexical entries on the basis

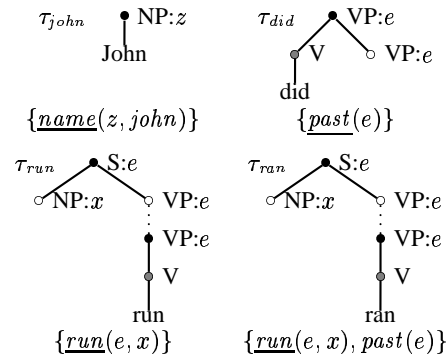


Figure 4: Example grammar

of their whole semantics, we select them on the basis of their index. That is, a lexical entry is selected iff its head unifies with a proposition in the input semantics. To preserve coherence, we further maintain the additional constraint that the total semantics of each selected entries subsumes (part of) the input semantics. For instance, given the grammar in Figure 4 (where semantic heads are underlined) and the input semantics  $run(e, x)$ ,  $name(x, john)$ ,  $past(e)$ , the generating matrix will be:

<u><math>name(x, john)</math></u>	$\tau_{john}$	<u><math>true</math></u>
<u><math>run(e, x)</math></u>	$\tau_{run}$	$\tau_{ran}$
<u><math>past(e)</math></u>	$\tau_{did}$	<u><math>true</math></u>

Given this matrix, two solutions will be found: the saturated tree for “John ran” satisfying the conjunction  $\tau_{john} \wedge \tau_{ran}$  and that for “John did run” satisfying  $\tau_{john} \wedge \tau_{run} \wedge \tau_{did}$ . No other solution is found as for any other conjunction of descriptions made available by the matrix, no saturated model exists.

## 5 Comparison with related work

Our generator presents three main characteristics: (i) It is based on an axiomatic rather than a generative view of grammar, (ii) it uses a TAG-like grammar in which the basic linguistic units are trees rather than categories and (iii) it assumes a flat semantics.

In what follows we show that this combination of features results in a generator which integrates the positive aspects of both top-down and bottom-up generators. In this sense, it is not unlike (Shieber et al., 1990)’s semantic-head-driven generation. As will become clear in the following section however, it differs from it in that it

integrates stronger lexicalist (i.e. bottom-up) information.

## 5.1 Bottom-Up Generation

Bottom-up or “lexically-driven” generators (e.g., (Shieber, 1988; Whitelock, 1992; Kay, 1996; Carroll et al., 1999)) start from a bag of lexical items with instantiated semantics and generates a syntactic tree by applying grammar rules whose right hand side matches a sequence of phrases in the current input.

There are two known disadvantages to bottom-up generators. On the one hand, they require that the grammar be semantically monotonic that is, that the semantics of each daughter in a rule subsumes some portion of the mother semantics. On the other hand, they are often overly non-deterministic (though see (Carroll et al., 1999) for an exception). We now show how these problems are dealt with in the present algorithm.

**Non-determinism.** Two main sources of non-determinism affect the performance of bottom-up generators: the lack of an indexing scheme and the presence of intersective modifiers.

In (Shieber, 1988), a chart-based bottom-up generator is presented which is devoid of an indexing scheme: all word edges leave and enter the same vertex and as a result, interactions must be considered explicitly between new edges and all edges currently in the chart. The standard solution to this problem (cf. (Kay, 1996)) is to index edges with semantic indices (for instance, the edge with category  $N/x:\text{dog}(x)$  will be indexed with  $x$ ) and to restrict edge combination to these edges which have compatible indices. Specifically, an active edge with category  $A(\dots)/C(c \dots)$  (with  $c$  the semantics index of the missing component) is restricted to combine with inactive edges with category  $C(c \dots)$ , and *vice versa*.

Although our generator does not make use of a chart, the constraint-based processing model described in (Duchier and Thater, 1999) imposes a similar restriction on possible combinations as it in essence requires that only these nodes pairs be tried for identification which (i) have opposite polarity and (ii) are labeled with the same semantic index.

Let us now turn to the second known source of non-determinism for bottom-up generators

namely, intersective modifiers. Within a constructive approach to lexicalist generation, the number of structures (edges or phrases) built when generating a phrase with  $n$  intersective modifiers is  $2^n$  in the case where the grammar imposes a single linear ordering of these modifiers. For instance, when generating “The fierce little black cat”, a naive constructive approach will also build the subphrases (1) only to find that these cannot be part of the output as they do not exhaust the input semantics.

- (1) The fierce black cat, The fierce little cat, The little black cat, The black cat, The fierce cat, The little cat, The cat.

To remedy this shortcoming, various heuristics and parsing strategies have been proposed. (Brew, 1992) combines a constraint-propagation mechanism with a shift-reduce generator, propagating constraints after every reduction step. (Carroll et al., 1999) advocate a two-step generation algorithm in which first, the basic structure of the sentence is generated and second, intersective modifiers are adjoined in. And (Poznanski et al., 1995) make use of a tree reconstruction method which incrementally improves the syntactic tree until it is accepted by the grammar. In effect, the constraint-based encoding of the axiomatic view of generation proposed here takes advantage of Brew’s observation that constraint propagation can be very effective in pruning the search space involved in the generation process.

In constraint programming, the solutions to a constraint satisfaction problem (CSP) are found by alternating propagation with distribution steps. Propagation is a process of deterministic inference which fills out the consequences of a given choice by removing all the variable values which can be inferred to be inconsistent with the problem constraint while distribution is a search process which enumerates possible values for the problem variables. By specifying global properties of the output and letting constraint propagation fill out the consequences of a choice, situations in which no suitable trees can be built can be detected early. Specifically, the global constraint stating that the semantics of a solution tree must be identical with the goal semantics rules out the generation of the phrases in (1b). In practice, we observe that constraint propagation is indeed very

efficient at pruning the search space. As table 5 shows, the number of choice points (for these specific examples) augments very slowly with the size of the input.

**Semantic monotonicity.** Lexical lookup only returns these categories in the grammar whose semantics subsumes some portion of the input semantics. Therefore if some grammar rule involves a daughter category whose semantics is not part of the mother semantics i.e. if the grammar is semantically non-monotonic, this rule will never be applied even though it might need to be. Here is an example. Suppose the grammar contains the following rule (where X/Y abbreviates a category with part-of-speech X and semantics Y):

$$\text{vp/call\_up}(X,Y) \rightarrow \text{v/call\_up}(X,Y), \text{np/Y}, \text{pp/up}$$

And suppose the input semantics is *call\_up(john, mary)*. On the basis of this input, lexical lookup will return the categories *V/call\_up(john,mary)*, *NP/john* and *NP/mary* (because their semantics subsumes some portion of the input semantics) but not the category *PP/up*. Hence the sentence “John called Mary up” will fail to be generated.

In short, the semantic monotonicity constraint makes the generation of collocations and idioms problematic. Here again the extended domain of locality provided by TAG is useful as it means that the basic units are trees rather than categories. Furthermore, as argued in (Abeillé and Schabes, 1989), these trees can have multiple lexical anchors. As in the case of vestigial semantics discussed in Section 4 above, this means that phonological material can be generated without its semantics necessarily being part of the input.

## 5.2 Top-Down Generation

As shown in detail in (Shieber et al., 1990), top-down generators can fail to terminate on certain grammars because they lack the lexical information necessary for their well-foundedness. A simple example involves the following grammar fragment:

- r1.  $s/S \rightarrow \text{np/NP}, \text{vp(NP)/S}$
- r2.  $\text{np/NP} \rightarrow \text{det(N)/NP}, \text{n/N}$
- r3.  $\text{det(N)/NP} \rightarrow \text{np/NP0}, \text{poss(NP0,NP)/NP}$
- r4.  $\text{np/john} \rightarrow \text{john}$
- r5.  $\text{poss(NP0,NP)/mod(N,NP0)} \rightarrow \text{s}$
- r6.  $\text{n/father} \rightarrow \text{father}$
- r7.  $\text{vp(NP)/left(NP)} \rightarrow \text{left}$

Given a top-down regime proceeding depth-first, left-to-right through the search space defined by the grammar rules, termination may fail to occur as the intermediate goal semantics NP (in the second rule) is uninstantiated and permits an infinite loop by iterative applications of rules r2 and r3.

Such non-termination problems do not arise for the present algorithm as it is lexically driven. So for instance given the corresponding DG fragment for the above grammar and the input semantics  $\{\text{left}(e, x), \text{father}(x, y), \text{name}(y, \text{john})\}$ , the generator will simply select the tree descriptions for “left”, “John”, “s” and “father” and generate the saturated model satisfying the conjunction of these descriptions.

## 6 Implementation

The ideas presented here have been implemented using the concurrent constraint programming language Oz (Smolka, 1995). The implementation includes a model generator for the tree logic presented in section 2, two lexical lookup modules (one for parsing, one for generation) and a small DG fragment for English which has been tested in parsing and generation mode on a small set of English sentences.

This implementation can be seen as a proof of concept for the ideas presented in this paper: it shows how a constraint-based encoding of the type of global constraints suggested by an axiomatic view of grammar can help reduce non-determinism (few choice points cf. table 5) but performance decreases rapidly with the length of the input and it remains a matter for further research how efficiency can be improved to scale up to bigger sentences and larger grammars.

## 7 Conclusion

We have shown that modulo some minor changes, the constraint-based approach to parsing presented in (Duchier and Thater, 1999) could also be used for generation. Furthermore, we have argued that the resulting generator, when combined with a TAG-like grammar and a flat semantics, had some interesting features: it exhibits the lexicalist aspects of bottom-up approaches thereby avoiding the non-termination problems connected with top-down approaches; it includes enough

Example	CP	Time
The cat likes a fox	1	1.2s
The little brown cat likes a yellow fox	2	1.8s
The fierce little brown cat likes a yellow fox	2	5.5s
The fierce little brown cat likes a tame yellow fox	3	8.0s

Figure 5: Examples

top-down guidance from the TAG trees to avoid typical bottom-up shortcomings such as the requirement for grammar semantic monotonicity and by implementing an axiomatic view of grammar, it supports a near-deterministic treatment of intersective modifiers.

It would be interesting to see whether other axiomatic constraint-based treatments of grammar could be used to support both parsing and generation. In particular, we intend to investigate whether the dependency grammar presented in (Duchier, 1999), once equipped with a semantics, could be used not only for parsing but also for generating. And similarly, whether the description based treatment of discourse parsing sketched in (Duchier and Gardent, 2001) could be used to generate discourse.

## References

- A. Abeillé and Y. Schabes. 1989. Parsing idioms in lexicalised TAGs. In *Proceedings of EACL '89*, Manchester, UK.
- R. Backofen, J. Rogers, and K. Vijay-Shanker. 1995. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language and Information*, 4(1).
- C. Brew. 1992. Letting the cat out of the bag: Generation for shake-and-bake MT. In *Proceedings of COLING '92*, Nantes, France.
- J. Carroll, A. Copestake, D. Flickinger, and V. Paznański. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of EWNLG '99*.
- A. Copestake, D. Flickinger, I. Sag, and C. Pollard. 1999. Minimal Recursion Semantics: An introduction. URL: <http://www-csli.stanford.edu/~aac/papers.html>, September.
- T. Cornell and J. Rogers. To appear. Model theoretic syntax. In L. Cheng and R. Sybesma, editors, *The GLOT International State of the Article Book 1*. Holland Academic Graphics, The Hague.
- D. Duchier and C. Gardent. 1999. A constraint-based treatment of descriptions. In H.C. Bunt and E.G.C. Thijsse, editors, *Proceedings of IWCS-3*, Tilburg.
- D. Duchier and C. Gardent. 2001. Tree descriptions, constraints and incrementality. In *Computing Meaning*, volume 2 of *Studies in Linguistics and Philosophy Series*. Kluwer Academic Publishers.
- D. Duchier and S. Thater. 1999. Parsing with tree descriptions: a constraint-based approach. In *NLULP'99*, Las Cruces, New Mexico.
- D. Duchier. 1999. Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on Mathematics of Language*, Orlando, Florida.
- A. Joshi. 1987. The relevance of Tree Adjoining Grammar to generation. In *Natural Language Generation*, chapter 16. Martinus Nijhoff Publishers, Dordrecht, Holland.
- L. Kallmeyer. 1999. *Tree Description Grammars and Underspecified Representations*. Ph.D. thesis, Universität Tübingen.
- M. Kay. 1996. Chart generation. In *Proceedings of ACL'96*, Santa Cruz, USA.
- J. D. McCawley. 1979. *Adverbs, Vowels, and other objects of Wonder*. University of Chicago Press, Chicago, Illinois.
- G. Perrier. 2000. Interaction grammars. In *Proceedings of 18th International Conference on Computational Linguistics (COLING 2000)*.
- V. Poznanski, J. L. Beaven, and P. Whitelock. 1995. An efficient generation algorithm for lexicalist MT. In *Proceedings of ACL '95*.
- O. Rambow, K. Vijay-Shanker, and D. Weir. 1995. D-tree Grammars. In *Proceedings of ACL '95*.
- S. Shieber, F. Pereira, G. van Noord, and R. Moore. 1990. Semantic-head-driven generation. *Computational Linguistics*, 16(1).
- S. Shieber. 1988. A Uniform Architecture for Parsing and Generation. In *Proceedings of ACL '88*.
- G. Smolka. 1995. The Oz Programming Model. In *Computer Science Today*, volume 1000 of LNCS.
- M. Stone and C. Doran. 1997. Sentence planning as description using Tree-Adjoining Grammar. In *Proceedings of ACL '97*.
- K. Vijay-Shanker. 1992. Using descriptions of trees in Tree Adjoining Grammars. *Computational Linguistics*, 18(4):481–518.
- P. Whitelock. 1992. Shake-and-bake translation. In *Proceedings of COLING '92*, Nantes, France.