Using Regular Tree Grammars to enhance Sentence Realisation

Claire Gardent, Benjamin Gottesman, Laura Perez-Beltrachini

Natural Language Engineering **1** (1): 1–**??**. Printed in the United Kingdom

Using Regular Tree Grammars to enhance Sentence Realisation

CLAIRE GARDENT
CNRS/LORIA, 615 Rue du jardin botanique, 54600 Villers-lès-Nancy, France

BENJAMIN GOTTESMAN
acrolinx GmbH, Rosenstraße 2, 10178 Berlin, Germany

LAURA PEREZ-BELTRACHINI
Nancy 1/LORIA, 615 Rue du jardin botanique, 54600 Villers-lès-Nancy, France.

(*Received 14 June 2010; revised 28 October 2010*)

Abstract

Feature-based regular tree grammars (FRTG) can be used to generate the derivation trees of a feature-based tree adjoining grammar (FTAG). We make use of this fact to specify and implement both an FTAG-based sentence realiser and a benchmark generator for this realiser. We argue furthermore that the FRTG encoding enables us to improve on other proposals based on a grammar of TAG derivation trees in several ways. It preserves the compositional semantics that can be encoded in feature-based TAGs; it increases efficiency and restricts overgeneration; and it provides a uniform resource for generation, benchmark construction, and parsing.

1 Introduction

Tree Adjoining Grammar (TAG, Joshi and Schabes (1997)) is a tree grammar formalism designed to describe natural languages. Each sentence derivation in a TAG yields both a *derived tree* representing the phrase structure of the sentence and a *derivation tree* specifying how the elementary TAG trees used to build this derived tree were combined. Interestingly, the derivation trees generated by TAGs form a regular tree language (Vijay-Shanker, Weir, and Joshi, 1987). Furthermore, TAG derivation trees have been shown to provide an intermediate representation from which both a sentence and its semantic representation can be derived (De Groote, 2002; Pogodalla, 2004; Shieber, 2006; Kanazawa, 2007). In other words, TAG derivation trees provide a pivot language which supports both parsing (going from a sentence to its possible syntactic structures and semantic representations) and generation (going from a semantic representation to one or more sentences).

In this paper, we argue that using a feature-based regular tree grammar (FRTG, Schmitz and Le Roux (2008)) encoding of a feature-based TAG (FTAG, Vijay-Shanker and Joshi (1988)) permits optimising and simplifying the processing of FTAG. We focus on sentence realisation (rather than parsing) and use an FTAG extended with a unification-based compositional semantics which permits associating with each sentence generated by the

grammar not only a syntactic structure but also a semantic representation. Making use of the translation from FTAG to FRTG defined by Schmitz and Le Roux (2008), we start by presenting two ways in which an FRTG encoding of this FTAG supports sentence generation, namely, (i) using this encoding to define a surface realiser and (ii) using it to derive a definite clause grammar (DCG) which can be used to automatically produce graduated, controlled sets of semantic representations (benchmarks) on which to test, compare, and optimise this surface realiser. Next, we compare our proposal with relevant work and explain why FRTG provides an interesting framework for FTAG-based surface realisation. In particular, we point out that the FRTG approach exhibits the following characteristics: an accurate treatment of the syntax/semantics interface; better management of time, space, and overgeneration than other approaches which have been proposed for FTAG-based surface realisation using derivation rather than derived trees; and a uniform resource for parsing, generation, and benchmark construction.

The paper is structured as follows. Section 2 provides the necessary background for the paper. It introduces FTAG, describes the specific grammar we use for our experiment (namely, SEMXTAG) and summarises the translation from FTAG to FRTG defined by Schmitz and Le Roux (2008). In Section 3, we present GENSEM, a tool for automatically producing graduated benchmarks for sentence generation. In Section 4, we describe a sentence generation algorithm based on the translation of SEMXTAG to FRTG. Finally, Section 5 spells out the three main motivations underlying the use of FRTG as a means to support FTAG-based surface realisation.

2 Grammars

We introduce FTAG, describe SEMXTAG, the specific grammar we use for our experiment, and summarise the FTAG-to-FRTG translation proposed by Schmitz and Le Roux (2008).

2.1 FTAG

A tree adjoining grammar is a tuple $\langle \Sigma, N, I, A, S \rangle$ with Σ a set of terminals, N a set of non-terminals, I a finite set of initial trees, A a finite set of auxiliary trees, and S a distinguished non-terminal ($S \in N$). Initial trees are trees whose leaves are labeled with substitution nodes (marked with a downarrow) or terminal categories. Auxiliary trees are distinguished by a foot node (marked with a star) whose category must be the same as that of the root node.

Two tree-composition operations are used to combine trees: substitution and adjunction. Substitution inserts a tree onto a substitution node of some other tree while adjunction inserts an auxiliary tree into a tree. In an FTAG, of which an example is given in Figure 1, the tree nodes are furthermore decorated with two feature structures (called top and bottom) which are unified during derivation as follows. On substitution, the top of the substitution node is unified with the top of the root node of the tree being substituted in. On adjunction, the top of the root of the auxiliary tree is unified with the top of the node where adjunction takes place; and the bottom features of the foot node are unified with the bottom features of this node. At the end of a derivation, the top and bottom of all nodes in the derived tree are unified. FTAG feature structures are non-recursive and consist of sets of feature/value

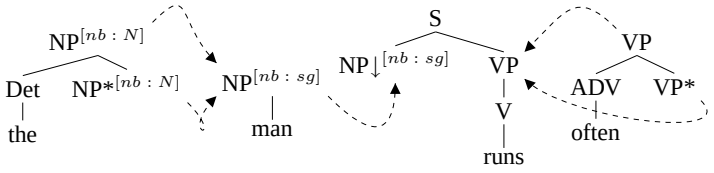


Fig. 1. Example feature-based tree adjoining grammar (N is a unification variable, sg is a constant, and $[f : v]$ is a feature structure with feature f and feature value v).

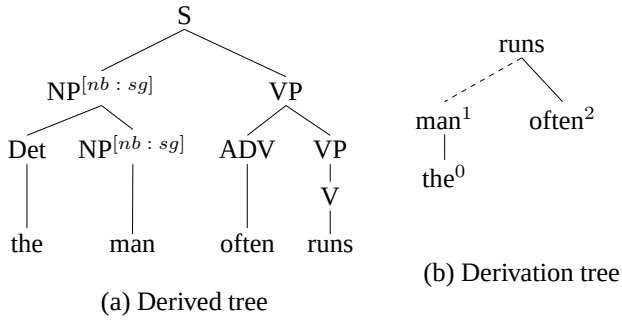


Fig. 2. Parse trees for "The man often runs" using the grammar of Figure 1. In the derivation tree, plain lines indicate adjunction and dotted ones substitution. For simplicity, tree names are replaced with the lemmas anchoring each elementary tree. The number on the upper right of each tree name gives the Gorn address of the node onto which the tree was inserted.

pairs where the value is either a constant or a unification variable. Unification variables can furthermore be coreferenced with any other value occurring in the same elementary tree.

In an FTAG, each sentence derivation is associated with both a derived tree representing the phrase structure of the sentence and a derivation tree recording how the corresponding elementary trees were combined to form the derived tree (Figure 2). Nodes in a derivation tree are labelled with the name of a TAG elementary tree. Edges are labelled with a description of the operation used to combine the TAG trees whose names label the edge vertices.

2.2 FTAG with semantics

To associate semantic representations with natural language expressions, the FTAG is modified as proposed by Gardent and Kallmeyer (2003). Each elementary tree is associated with a flat semantic representation. For instance, in Figure 3, the trees for *mary* and *run* are associated with the semantics $l_6 : mary(x_6)$ and $l_7 : run(e_7, x_7)$, respectively. Importantly, the arguments of a semantic functor are represented by unification variables which occur both in the semantic representation of this functor and on some nodes of the associated syntactic tree. For instance, in Figure 3, the semantic index x_7 occurring in the semantic representation of *run* also occurs on the subject substitution node of the associated elementary tree. The value of semantic arguments is determined by the unifications resulting from adjunction and substitution. For instance, the semantic index x_7 in the tree

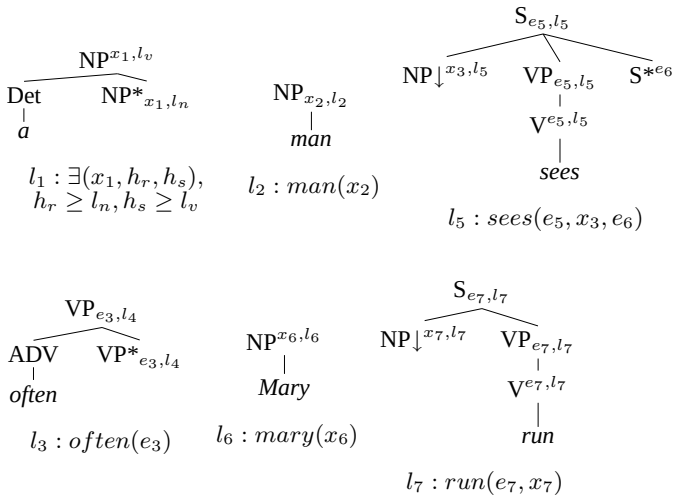


Fig. 3. An FTAG augmented with a unification-based semantics. For the sake of clarity, feature structures are abbreviated, feature percolation has been simplified precluding the possibility that adjunction modifies feature values and only the semantic feature values relevant for semantic construction are indicated. C^{x,l}/C$_{x,l}$ abbreviate a node with category C and a top/bottom feature structure including the feature-value pairs { index:x, label:l}.

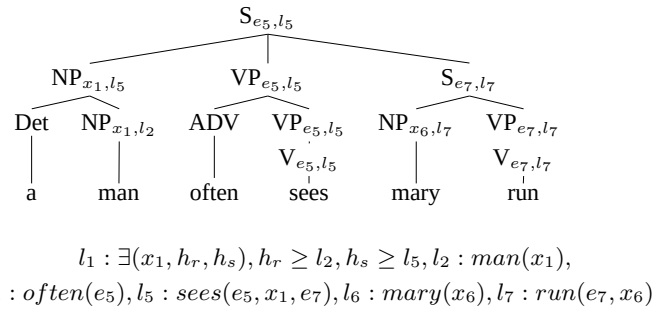


Fig. 4. Derived tree and semantics for "A man often sees Mary run"

for *run* is unified during substitution with the semantic index labelling the root node of the tree for *mary*. As a result, the semantics of *Mary run* is $\{l_6 : mary(x_6), l_7 : run(e_7, x_6)\}$.

The semantic representation language used (L_U) is a unification-based language which describes first order formulae in the sense that the model of a given L_U formula is a set of first order formulae (Gardent and Kallmeyer, 2003). For instance, the formula in Figure 4 describes the first order formula

$$\exists x_1.(man(x_1) \wedge often(e_5) \wedge sees(e_5, x_1, e_7) \wedge mary(x_6) \wedge run(e_7, x_6))$$

More generally, L_U formulae are flat, underspecified FOL formulae. They are flat in that the recursive tree structure of a FOL formula is transformed into a conjunction of non-recursive labelled formulae whereby the label of each formula is used to indicate its position in the initial tree structure. They are underspecified in that the scope of scope-

bearing operators (quantifiers, modals, negation) is specified by underconstrained scoping constraints between so-called holes (written h, h_i) and labels (written l, l_i). Thus, the formulae of L_U are labelled elementary predications ($l : R^n(i_1, \ldots, i_n)$ with l a label constant, R^n an n-ary relation, and i_j variables over individuals and/or labels/hole constants), scoping constraints ($h \geq l$ with h a hole constant and l a label constant), and conjunctions (ϕ, ψ with ψ and ϕ formulae of L_U). The models described by L_U formulae are defined by the set of possible "pluggings", i.e., injections from the holes of a formula to the labels of this formula. The following example illustrates this. Suppose the sentence in (1) is assigned the L_U formula (2).

(1) Every dog chases a cat

(2) $l_0 : \forall(x, h_1, h_2), h_1 \geq l_1, l_1 : D(x), h_2 \geq l_2, l_2 : Ch(x, y), l_3 : \exists(x, h_3, h_4), h_3 \geq l_4, l_4 : C(y), h_4 \geq l_2$

Only two pluggings are possible for this formula in (2) namely $\{h_1 \rightarrow l_1, h_2 \rightarrow l_3, h_3 \rightarrow l_4, h_4 \rightarrow l_2\}$ and $\{h_1 \rightarrow l_1, h_2 \rightarrow l_2, h_3 \rightarrow l_4, h_4 \rightarrow l_0\}$. They yield the following meaning representations for (1):

(3) a. $l_0 : \forall(x, l_1, l_3), l_1 : D(x), l_2 : Ch(x, y), l_3 : \exists(x, l_4, l_2), l_4 : C(y)$
b. $l_0 : \forall(x, l_1, l_2), l_1 : D(x), l_2 : Ch(x, y), l_3 : \exists(x, l_4, l_0), l_4 : C(y)$

For more details on the interpretation of L_U and on the semantic representations it permits associating with a grammar of natural language, refer to Gardent and Kallmeyer (2003). The proposal described in this paper is, however, largely independent of the specific semantics used and only requires a tree adjoining grammar that is equipped with a unification-based semantics, such as SEMXTAG.

2.3 SemXTAG

SEMXTAG is an FTAG for English whose syntactic coverage approaches that of XTAG, the FTAG developed for English by the XTAG group (The XTAG Research Group, 2001). Like XTAG, it contains around 1300 elementary trees and covers auxiliaries, copula, raising and small clause constructions, topicalization, relative clauses, infinitives, gerunds, passives, adjuncts, ditransitives and datives, ergatives, it-clefts, wh-clefts, PRO constructions, noun-noun modification, extraposition, sentential adjuncts, imperatives, and resultatives. Unlike XTAG, SEMXTAG is augmented with a unification-based compositional semantics of the type described above, with which all of its syntactic constructs are associated.

2.4 Converting SemXTAG to FRTG

In this section, we summarise the FTAG to FRTG transformation of Schmitz and Le Roux (2008). We start by presenting the TAG to RTG conversion, that is, the conversion for a grammar without feature structures. We then go on to indicate how feature structures are converted. For a more precise description of this FTAG to FRTG conversion, we refer the reader to Schmitz and Le Roux (2008).

A *regular tree grammar* (Comon, Dauchet, Jacquemard, Lugiez, Tison, and Tommasi, 1997) is a grammar whose rules rewrite a non-terminal symbol as a tree whose internal

nodes are each labeled with a terminal symbol and whose leaf nodes are each labeled with a terminal or non-terminal symbol.

Formally, an RTG is a 4-tuple $G = (S, \mathcal{N}, F, R)$ consisting of an axiom S, a finite set $\mathcal{N}$ of zero-arity non-terminal symbols with $S \in \mathcal{N}$, a set $\mathcal{F}$ (disjoint with $\mathcal{N}$) of terminal symbols each having a fixed arity, and a finite set $\mathcal{R}$ of production rules of the form $A \rightarrow \beta$, with A a non-terminal of $\mathcal{N}$ and β a *term* over $\mathcal{F} \cup N$. A term over some set of fixed-arity symbols M is defined recursively as a symbol of M applied to n arguments with n equal to the arity of the symbol and each of the arguments being a term over M. A set of fixed-arity symbols is also called a *ranked alphabet*, and the set of terms over the ranked alphabet M is written $T(M)$.

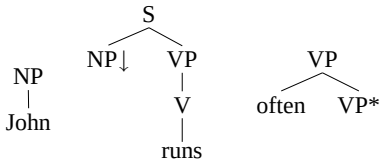
The language described by an RTG is a *regular tree language*. A given RTG $G = (S, \mathcal{N}, \mathcal{F}, \mathcal{R})$ describes the language consisting of all terms t over $\mathcal{F}$ such that the axiom S can be rewritten as t via a series of rewrites licensed by the rules in $\mathcal{R}$. In other words, to derive a term of the language, we start from the axiom and apply rules until we have a term containing no non-terminal symbols.

More formally, the derivation relation $\rightarrow_G$ associated to G is a relation on pairs of terms of $T(\mathcal{F} \cup \mathcal{N})$ such that $s \rightarrow_G t$ if and only if there is a rule $A \rightarrow \alpha \in \mathcal{R}$ such that substituting α for an instance of A in s gives t; and the language generated by G, denoted by $L(G)$, is $\{s \in T(\mathcal{F}) \mid S \rightarrow_G^+ s\}$ with $\rightarrow_G^+$ the transitive closure of $\rightarrow_G$. The subscript G on the symbols $\rightarrow_G$ and $\rightarrow_G^+$ can be omitted if the grammar is clear from the context.

As is well known (Vijay-Shanker, Weir, and Joshi, 1987; Shieber, 2006), RTG can be used to generate the derivation trees licensed by a TAG grammar. Intuitively, the RTG representation of a TAG elementary tree is a rule that rewrites the requirement satisfied by that tree as a local tree whose root is the tree name and whose leaves are the introduced requirements. A substitution / adjunction requirement for a tree of root category X is written as X_S and X_A, respectively.

Figure 5 shows the rules of an example RTG that describes the derivation trees of the toy TAG grammar depicted in the left part of the figure. The RTG terminals ({ *john, runs, often,* ϵ}) refer to the elementary trees of the TAG grammar while its non-terminals ($\{NP_S, S_S, NP_A, VP_A, V_A, S_A\}$) describe the adjunction and substitution requirements that can be introduced by an elementary tree. Further, each elementary tree t in the input TAG gives rise to an RTG rule whose left hand side (lhs) expresses the syntactic requirement that t can satisfy and whose right hand side (rhs) expresses the syntactic requirements it introduces. If the tree is an auxiliary tree, it can satisfy an adjunction requirement and the category labelling the lhs of the RTG rule is subscripted with A. If it is an initial tree, the lhs category of the RTG rule is subscripted with S to indicate that it can satisfy a substitution requirement. Further, each node in the elementary tree that either requires a substitution or allows for an adjunction introduces a daughter node in the rhs RTG term whose category reflects the allowed/required adjunction/substitution. To capture the fact that adjunction is optional, there are additional rules allowing any adjunction requirement to be rewritten as the symbol ϵ, a terminal symbol of the RTG.

We just saw how to map a TAG to an RTG of TAG derivations. Schmitz and Le Roux (2008) further extend this mapping to FTAG as follows. In the resulting FRTG, each non-terminal symbol on the left and right side of a rule is marked up with a feature structure with top and bottom attributes. For a symbol on the right side, the values of those attributes



$$
\begin{array}{lrcl}
\text{r1.} & NP_S & \rightarrow & john(NP_A) \\
\text{r2.} & S_S & \rightarrow & runs(S_A\ NP_S\ VP_A\ V_A) \\
\text{r3.} & VP_A & \rightarrow & often(VP_A) \\
\text{r4.} & NP_A & \rightarrow & \epsilon \\
\text{r5.} & S_A & \rightarrow & \epsilon \\
\text{r6.} & V_A & \rightarrow & \epsilon \\
\text{r7.} & VP_A & \rightarrow & \epsilon
\end{array}
$$

Fig. 5. Example RTG describing the derivation trees of a toy TAG.

are equal to the top and bottom feature structures of the corresponding TAG tree node (substitution node or adjunction site). For the symbol on the left, they are the *interface* of the tree to any node into which the tree is inserted. When an initial tree is inserted into a substitution node of another tree, its root node's top unifies with the substitution node's top. Thus, the interface of the initial tree is its root node's top, and this appears as the top attribute of the symbol on the left side of the corresponding RTG rule. For an auxiliary tree, the interface is the top of the root node and the bottom of the foot node (cf. Section 2.1), so these appear as the top and bottom, respectively, of the left side of the corresponding rule.

There will generally be co-indexed feature values in a rule. In a substitution rule such as (4), which is a translation of the 'Mary' tree from Figure 3, the top value of the left side symbol is equal to the top feature structure of the root node of the tree, and therefore is co-indexed with the top value of the right side symbol that embodies that node. In an epsilon rule such as (5), the top and bottom values on the left side are co-indexed with each other to enforce the requirement that the top and bottom feature structures of each node in the derived tree must unify.

(4)

$$NP_{S\left[\textbf{top}\ \boxed{1}\right]} \rightarrow Mary(NP_{A\left[\textbf{top}\ \boxed{1}\left[\text{index}\ \ x_6\right]\right]})$$

(5)

$$NP_{A\begin{bmatrix}\textbf{top} & \boxed{1}\\ \textbf{bottom} & \boxed{1}\end{bmatrix}} \rightarrow \epsilon$$

3 Constructing benchmarks for sentence generation

Unlike parsing, where the input (strings) can be taken from existing text, sentence generation requires abstract input data that is not readily available. We present an approach for producing test input for a sentence realiser which draws on that used by Nederhof (1996) and Purdom (1972) for testing a parser. This approach consists of traversing the grammar to produce the semantic formulae it associates with sentences of the language it describes. Using semantic representations generated from the grammar ensures that the representations given to the sentence realiser are in the language defined by this grammar. Hence a generation failure necessarily indicates a flaw in the generator's design as opposed to a lack of coverage by the grammar. Furthermore, this permits focusing on test cases that can be handled by the grammar and testing the generator's performance. We show that by deriving a definite clause grammar (DCG) from SEMXTAG, using as a pivot grammar the FRTG of FTAG derivations described in the previous section, we can create graduated benchmarks

$$
\begin{array}{rcl}
NP_S & \rightarrow & john\ NP_A \\
S_S & \rightarrow & runs\ S_A\ NP_S\ VP_A\ V_A \\
VP_A & \rightarrow & often\ VP_A \\
NP_A & \rightarrow & \epsilon \\
S_A & \rightarrow & \epsilon \\
V_A & \rightarrow & \epsilon \\
VP_A & \rightarrow & \epsilon
\end{array}
$$

Fig. 6. Rules of a CFG derived from an RTG of TAG derivations

for sentence generation that permit testing performance on cases of varying computational complexity. We concentrate here on describing our approach to formula generation, i.e. GENSEM. For details about how to use GENSEM to produce tailored test-suites, test-suites examples and a practical case of test-suites usage, we refer the reader to Gardent, Gottesman, and Perez-Beltrachini (2010).

We begin by showing in Section 3.1 how to automatically derive a DCG from the FRTG of FTAG derivations described in the previous section. We then show how the resulting DCG permits generating formulae while enabling control over the set of semantic representations to be produced (Section 3.2).

3.1 Converting SemXTAG to a DCG

In the DCG formalism, a grammar is represented as a set of Prolog definite clauses, and Prolog's query mechanism provides built-in grammar traversal. We take advantage of this by deriving a DCG from SEMXTAG and then using Prolog queries to generate semantic representations that are licenced by SEMXTAG.

Our algorithm for converting SEMXTAG to a DCG proceeds in two steps. We first derive an FRTG of SEMXTAG's derivations as described in the previous section, then convert this FRTG to a DCG. We will show that there exists a mapping from our FRTG to a DCG such that the trees in the language described by the FRTG are in a one-to-one relationship with, and can easily be reconstructed from, the derivations of the DCG. As before, we start by explaining the mapping between featureless versions of the grammars, then explain how feature structures are handled.

Any RTG can be converted to a CFG (Gécseg and Steinby, 1997), which is what motivates our use of RTG as an intermediate representation. In the case of an RTG of TAG derivations, each of whose rules has a local tree on the right-hand-side, this can be done simply by flattening each right-hand-side tree into a list consisting of its root node followed by its leaves. If we apply this operation, for example, to the rules of the RTG of derivations from Figure 5, the result, shown in Figure 6, forms the rule set of a CFG.

Figure 7 illustrates the mapping between the trees generated by the RTG and those generated by the CFG. Reconstructing the former from the latter is trivial. It suffices to relabel each internal node in the derivation tree with the label of the unique leaf node among its daughters, and then prune the leaf nodes. This effectively reverses the flattening of the rules' right-hand sides.

Extending the RTG to CFG mapping with feature structures is straightforward. In fact,

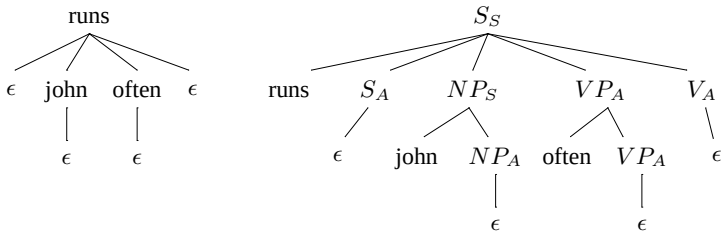


Fig. 7. An RTG-generated tree and corresponding CFG derivation

the procedure for mapping an FRTG rule to a CFG rule is exactly the same except that the CFG non-terminals are now complex non-terminal categories including a feature structure.

Now, as definite clause grammars are essentially a superclass of feature-augmented CFGs, formulating an instance of the latter as the former is straightforward. It is primarily a matter of writing down the rules in the particular Prolog syntax of DCGs. For details on the DCG implementation we refer the reader to Gottesman (2009).

Each FRTG rule other than the epsilon rules is additionally associated with a semantic formula containing unification variables. These were carried over as is from the TAG trees to the FRTG rules, and we carry them over to the DCG rules as well.

Figures 8, 9, and 10 illustrate the TAG to DCG conversion process we just sketched. Figure 8 shows a toy FTAG. For each tree, the anchor family and semantics are shown underneath it. The anchor node[1] is indicated with a diamond ($\diamond$). This grammar fragment includes only two features, namely the semantic label and the index. Figure 9 shows the result of translating this fragment to an FRTG and Figure 10 shows the result of converting this FRTG to a DCG , with a minimal lexicon added. Note that all non-terminals are represented using the `rule` predicate. The (non-hidden) arguments of the basic rule predicate[2] are `rule(Category,Subscript,Top,Bottom,Semantics)`. Its first two arguments encode the base name and subscript (init = initial/substitution = subscript S, aux = auxiliary/adjunction = subscript A) of the non-terminal symbol. The top and bottom features are represented by the third and fourth arguments, respectively. The fifth argument is the semantics slot.

3.2 Querying the DCG

We now show how to run a query that will cause Prolog to traverse the DCG encoding of the SEMXTAG grammar so as to find a valid derivation tree and produce a semantic formula as output.

We define a Prolog predicate for querying against the DCG, as follows. Its one input argument, `Cat`, is the label of the root node of the derivation tree (typically `s`), and its one output argument, `Sem`, is the semantic representation associated with that tree[3].

[1] The anchor node is the node in an elementary TAG tree which immediately dominates the word lexicalising that tree.
[2] "basic" because we will augment the DCG rules later.
[3] The 6th and 7th arguments of the rule call are the hidden arguments needed by the DCG.

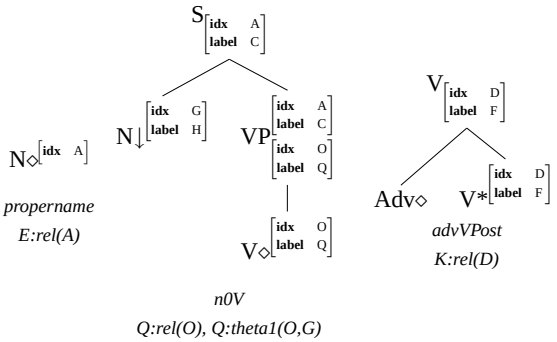


Fig. 8. TAG fragment

$$N_S[\text{top Top}] \rightarrow propername(N_A[\text{top Top}, \text{top}[\text{idx A}]])$$
E:rel(A)

$$S_S[\text{top Top}] \rightarrow n0V(S_A[\text{top Top}, \text{bottom}[\text{idx A}, \text{label C}]]\ N_S[\text{top}[\text{idx G}, \text{label H}]]\ VP_A[\text{top}[\text{idx A}, \text{label C}], \text{bottom}[\text{idx O}, \text{label Q}]]\ V_A[\text{top}[\text{idx O}, \text{label Q}]])$$
Q:rel(O), Q:theta1(O,G)

$$V_A[\text{top Top}] \rightarrow advVPost(V_A[\text{top Top}, \text{bottom}[\text{idx D}, \text{label F}]]\ Adv_A)$$
K:rel(D)

$$X_A[\text{top F}, \text{bottom F}] \rightarrow \epsilon$$

Fig. 9. FRTG translation of TAG fragment

```
genSem(Cat,Sem) :-
    rule(Cat,init,_,_,Sem,_,[]).
```

In order to give the users some control over the sorts of semantic representations that they get back from a query against the DCG, we augment the DCG in such a way as to allow control over the TAG family[4] of the root tree in the derivation tree, over the number and type of adjunctions in the derivation, and over the depth of substitutions. To implement control over the family is quite simple: we need merely to index the DCG rules by family and modify the GENSEM call accordingly. For instance, the `s` DCG rule from Figure 10 becomes :

```
rule(s,init,FsTop,_,n0V,Q:Formula1;...) -->
```

[4] TAG families group together trees which belong together, in particular, the trees associated with various realisations of a specific subcategorisation type. Thus, here the notion of TAG family is equivalent to that of subcategorisation type.

```
rule(_,aux,FS,FS,none) -->  [epsilon].    % epsilon rule

rule(n,init,FsTop,_,E:Formula1;N1Sem) -->
     [Lemma], {lexicon(Lemma,propername,[Rel])},
     rule(n,aux,FsTop,_,N1Sem),
     {FsTop = fs(A,_)},
     {Formula1 =.. [Rel,A]}.

rule(s,init,FsTop,_,
       Q:Formula1;Q:Formula2;S1Sem;N2Sem;VP3Sem;V4Sem) -->
     [Lemma], {lexicon(Lemma,n0V,[Rel,Theta1])},
     rule(s,aux,FsTop,fs(A,C),S1Sem),
     rule(n,init,fs(G,H),_,N2Sem),
     rule(vp,aux,fs(A,C),fs(O,Q),VP3Sem),
     rule(v,aux,fs(O,Q),_,V4Sem),
     {Formula1 =.. [Rel,O]},
     {Formula2 =.. [Theta1,O,G]}.

rule(v,aux,FsTop,_,K:Formula1;V1Sem;ADV2Sem) -->
     [Lemma], {lexicon(Lemma,advVPost,[Rel])},
     rule(v,aux,FsTop,fs(D,F),V1Sem),
     rule(adv,aux,_,_,ADV2Sem),
     {Formula1 =.. [Rel,D]}.

lexicon(john,propername,[john]).
lexicon(arrive,n0V,[arrive,agent]).
lexicon(also,advVPost,[also]).
```

Fig. 10. DCG translation of FRTG fragment.

```
     [Lemma],
     {lexicon(Lemma,n0V,[Rel,Theta1])},
     ...
```

We permit restrictions on adjunctions by adding an additional argument to the grammar symbols, namely a vector of non-negative integers representing the number of non-null adjunctions of each type that are in the derivation subtree dominated by the symbol. By 'type' of adjunction, we mean the category of the adjunction site. In DCG terms, a non-null adjunction of a category `X` is represented as the expansion of an `x/aux` symbol other than as ϵ. So, for example, a DCG symbol associated with the vector `[1,0,0,0,0]`, where the five dimensions of the vector correspond to the `n`, `np`, `v`, `vp`, and `s` categories, respectively, dominates a subtree containing exactly one `n/aux` symbol expanded by a non-epsilon rule, and no other `aux` symbol expanded by a non-epsilon rule.

To implement the functionality of the vector, we define a special predicate to handle the divvying up of a mother node's vector among the daughters. We link the vector associated with the root of the derivation to the query predicate.

Finally, we add an additional argument to the DCG rule and to the GENSEM call to control the traversal depth with respect to the number of substitutions applied. The overall depth of each derivation is therefore constrained both by the user-defined adjunction and substitution-depth constraints.

Our query predicate now has four input arguments and one output argument:

```
genSem(Cat,Fam,[N,NP,V,VP,S],Dth,Sem):-
     rule(Cat,init,_,_,Fam,
          [N,NP,V,VP,S],Dth,Sem,_,[]).
```

4 TAG-based sentence generation

We now present the sentence realiser RTGEN (Perez-Beltrachini, 2009), which is based on the FRTG encoding of SEMXTAG. RTGEN synthesises different techniques that have been shown in the past to improve sentence generation runtimes. We first sketch the basic algorithm. We then present three alternative ways of implementing RTGEN, which we will use to compare the FRTG approach with related work (cf. Section 5).

RTGen's base algorithm. In essence, RTGEN implements a chart-based Earley algorithm for the FRTG encoding of SEMXTAG. Table 1 sketches the algorithm. The standard item representation is the pair $[(A, d) \rightarrow T_a((B_1, d_1), .., \bullet(B_i, d_i), ..., (B_n, d_n)), \psi]$. In the first component, the dot in the production marks the point reached in the generation of the derivation tree. The non-terminal symbols (B_i, d_i) in the dotted rule are complex non-terminals from the FRTG rules (i.e. a non-terminal symbol, syntactic category and operation type, B_i and a feature structure d_i). T_a is the ranked terminal of the FRTG rule (i.e. is the elementary tree family). The second component of the item, ψ, is a flat semantic formula. In the items, we do not keep track of input string positions, as is usually done when parsing a string, but rather we keep the associated input semantic information. The algorithm starts from the initial fact, the axiom, $[S' \rightarrow \bullet S_S, \emptyset]$. Note that in this item the non-terminal symbol S_S is the axiom in the FRTG grammar while the second component represents the empty semantics. As we are generating from an input semantics (i.e. the semantic input to the realiser), the subset of the input semantics analysed so far is empty. On the other hand, in the goal item $[S' \rightarrow S_S \bullet, \phi]$ the dot at the end of the item production means that the whole derivation tree with root S_S has been traversed. At this point the semantics ϕ should be exactly the input semantics. This RTGEN algorithm includes mechanisms such as sharing and packing (Gardent and Perez-Beltrachini, 2010).

Table 1. *RTGen derivation tree generation algorithm.*

Axiom	$\overline{[S' \rightarrow \bullet S_S, \emptyset]}$
Goal	$[S' \rightarrow S_S \bullet, \phi]$ where ϕ is the input semantics.
Prediction	$\frac{[(A, d) \rightarrow T_a(\alpha \bullet (B, d_i)\beta), \varphi]}{[(B, \sigma(d')) \rightarrow T_b(\bullet(B_1, \sigma(d'_1)), ..., (B_n, \sigma(d'_n))), \psi]}$ where $(B, d') \rightarrow T_b((B_1, d'_1), ..., (B_n, d'_n))$ is a rule in the grammar with associated semantics ψ, $\sigma = mgu(d_i, d')$ and $\varphi \cap \psi = \emptyset$
Completion	$\frac{[(A, d) \rightarrow T_a(\alpha \bullet (B, d_i)\beta), \varphi][(B, d') \rightarrow T_b(\beta)\bullet, \varphi]}{[(A, \sigma(d)) \rightarrow T_a(\alpha(B, \sigma(d_i)) \bullet (C, \sigma(d_{i+1}))\delta), \phi]}$ where $\sigma = mgu(d_i, d')$, $\varphi \cap \psi = \emptyset$ and $\varphi \cup \psi = \phi$

Three ways to implement RTGen. Depending on how much linguistic information (in the form of feature constraints from the feature structures) is preserved in the FRTG rules, several RTGEN configurations can be tried out, each reflecting a different division of labour between constraint solving and structure building.

RTGen-all: all the feature structure information present in the SEMXTAG elementary trees is carried over to the FRTG rules.

RTGen-level0: only the syntactic category and the semantic features are preserved by the conversion.

RTGen-selective: uses exactly the subset of features that induce feature structure unification failures during generation. We identify this subset while running RTGen-all by (automatically) observing which features effectively block item combination.

5 Why use RTG?

The general motivation for using an FRTG transformation when generating with an FTAG is that, in contrast to other approaches that have used TAG derivation trees for generation (Koller and Striegnitz, 2002; Koller and Stone, 2007), the FRTG-based approach preserves all feature information, thereby providing an exact grammar of FTAG derivation trees. We now discuss three main benefits of the FRTG approach.

5.1 Preserving the syntax/semantics interface

Given some input semantics, surface realisation involves building a syntactic tree whose compositional semantics is the input semantics. When generating from the derivation tree, it is therefore necessary that derivations can be coupled with the appropriate compositional semantics. For standard FTAG derivation trees, however, this is known not to be the case (Frank and Genabith, 2001). Indeed, there are known cases of linguistic constructs (quantifiers, wh-questions, and raising verbs) where the standard FTAG derivation tree cannot be associated with the appropriate compositional semantics, in essence because two elementary trees that stand in a semantic relation are not in a mother-daughter relationship[5]. As shown by Gardent and Kallmeyer (2003) and sketched in Section 2, a simple solution to this quandary is to associate a compositional semantics with FTAG derived trees using features and feature structure unification. Since furthermore, the FTAG-to-RTG conversion preserves both the semantics and the feature structure information contained in the initial FTAG, it follows that the derivation trees built by RTGEN also support the appropriate semantics. This contrasts, in particular, with the approach described by Koller and Striegnitz (2002). In that approach, similarly to the RTGEN approach, surface realisation relies on an FTAG which is converted to a dependency grammar that models the contribution of each FTAG elementary tree to the derivation tree. However, since, in that approach, feature

[5] For instance, in an FTAG (such as SEMXTAG) where the determiner is an auxiliary tree, there will be no direct connection in the derivation tree of the sentence *Every yogi has a guru* between the universal quantifier licensed by the determiners (*Every* and *a*) and their scope (licensed by the verb *has*). For a detailed discussion of why featureless TAG derivation trees fail to support compositional semantics, see Frank and Genabith (2001) and Gardent and Kallmeyer (2003).

strategy	GF	chart	unpacked-chart	seconds
RTGen-all	15.05	918.31	2538.98	0.99
RTGen-level0	1118.06	2018	6898.28	1.41
RTGen-selective	27.08	910.34	2531.23	0.44

Table 2. RTGEN *average chart sizes and processing times on 618 test cases from a* GENSEM*-generated benchmark. Each test case has 3 modifications, distributed in various ways between adjectival and adverbial modifications and combined (in some cases) with lexical ambiguity, 10 trees for each modifier. The second column, Generation Forest (GF), is the number of derivation trees present in the generated parse forest. The third and fourth columns show the chart and unpacked chart sizes, respectively. The last column shows the runtime in seconds.*

structures are not taken into account, it is unclear how the derivation trees of the known problematic cases can be assigned an appropriate semantics and, consequently, how the corresponding sentences could be generated.

5.2 Efficiency and overgeneration

Real world grammars such as XTAG or SEMXTAG typically make heavy use of feature structures and feature unification to restrict the number of elementary trees and appropriately model linguistic phenomena such as, for instance, verb/subject agreement or, as mentioned in the preceding section, to appropriately model the syntax/semantic interface.

Importantly, the FRTG encoding provided by Schmitz and Le Roux (2008) preserves the feature structure information contained in SEMXTAG, thereby ensuring that only sentences generated by SEMXTAG are produced. Moreover, as mentioned in Section 4, RTGEN permits experimenting with three levels of feature information (RTGen-all, RTGen-level0, and RTGen-selective). To explore the impact of features on efficiency and overgeneration and to compare the FRTG approach with those of Koller and Striegnitz (2002) and Gardent and Kow (2007), we ran RTGEN in each of the three modes on a benchmark generated using the GENSEM benchmark generator described in Section 3. Table 2 presents an analysis of cases involving three modifiers. The results show that running RTGEN with no feature information (other than semantic features) leads to an increased chart size; to runtimes that are higher on average than for full sentence generation, that is, realisation using the full grammar with all constraints; and to massive overgeneration (on average, 1108.06 sentence trees produced when using no features against 15.05 when using all features). This suggests that using a filtering step based only on semantic indices is a poor strategy. Using GENSEM, we compared RTGEN with the GENI (Gardent and Kow, 2007) surface realiser, which uses such a filter. And indeed, we found that in terms of space, RTGEN consistently outperformed GENI (cf. (Gardent and Perez-Beltrachini, 2010)). Similarly, the approach of Koller and Striegnitz (2002) builds derivation trees without taking feature structures into account, thus raising the question of how the resulting massive overgeneration will impact efficiency once the postprocessing step needed to check feature information is taken into account.

5.3 Versatility

A third advantage of using FRTG as a means to describe the derivation trees of an FTAG is that, by maintaining a grammar-based approach, it permits benefiting from the tools and techniques developed for computational grammars. This contrasts with the planning-based approach of Koller and Stone (2007) in two ways.

First, sophisticated search strategies that were developed to handle the exponential complexity of the surface realisation problem can be drawn upon. As shown by Koller and Hoffmann (2010), the planning approach fails to scale to conjunctions of five basic clauses such as *The man greets the man and the man greets the man and the man greets the man and the man greets the man and the man greets the man*. For such cases, all planners and planner strategies tested by Koller and Hoffmann (2010) time out. The Earley-based search strategy used by RTGEN, in contrast, yields the expected sentence in 2.03 seconds of CPU time. More generally, while the planning approach is an interesting way of dealing with the interactions between surface realisation and the generation of referring expressions, the grammar-based approach seems better suited to handle "real world" surface realisation, i.e. the production of well-formed sentences of arbitrary length and complexity.

Second, the FRTG encoding of SEMXTAG can be put to work in different ways. In particular, we showed that it could be used either to produce the sentences verbalising a given meaning (surface realisation) or – by translating it to a DCG – to generate the semantic representations licenced by SEMXTAG and obeying a set of user defined constraints (benchmark generation). Furthermore, as Schmitz and Le Roux (2008) mentioned, the FRTG can also be used for parsing (going from string to meaning) provided it is extended with topological information as proposed by Kuhlmann (2010). In short, the FRTG approach provides a uniform resource which supports various types of processing. While the planning operators of Koller and Stone (2007) could in principle also be used in different ways, it remains to be seen exactly how one might adapt planning operators and planning strategies to effectively model parsing, surface realisation, and/or the construction of benchmarks.

6 Conclusion

In this paper, we exploited FRTG both to implement a sentence realiser and to derive a benchmark generator for sentence realisation. We argued that FRTG, because it fully preserves feature information, better supports completeness, efficiency, and versatility than other proposals that have been put forward for using TAG derivation trees as a basis for sentence generation.

Interestingly, Maxwell and Kaplan (1993) show that a more sophisticated approach to constraint solving and to its interleaving with chart processing renders the non-interleaved approach more effective than the interleaved one. We plan to examine whether this observation applies to SEMXTAG and RTGEN. Another interesting topic for further research is to explore to what extent GENSEM can be used to provide generic test-beds for surface realisation, i.e. test-beds that can be exploited by realisers based on grammars other than SEMXTAG.

Acknowledgments

We are grateful to Sylvain Schmitz for providing us with the FTAG-to-FRTG conversion tool. We also thank the reviewers for their helpful comments.

References

Comon, H., M. Dauchet, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi (1997). Tree automata techniques and applications. Technical report.

De Groote, P. (2002, May). Tree Adjoining Grammars as Abstract Categorial Grammars. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, Venice, Italy, pp. 145–150.

Frank, A. and J. Genabith (2001, June). GlueTag. Linear Logic based Semantics for LTAG. In M. Butt and T. H. King (Eds.), *Proceedings of the LFG01 Conference*, University of Hong Kong, Hong Kong, pp. 104–126. CSLI Publications, Stanford, California, USA. ISSN 1098-6782.

Gardent, C., B. Gottesman, and L. Perez-Beltrachini (2010, August). Comparing the performance of two TAG-based Surface Realisers using controlled Grammar Traversal. In *Proceedings of Coling 2010 (Poster session)*, Beijing, China, pp. 338–346. Coling 2010 Organizing Committee.

Gardent, C. and L. Kallmeyer (2003, April). Semantic construction in Feature-Based TAG. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, Budapest, Hungary, pp. 123–130. Association for Computational Linguistics, Morristown, NJ, USA.

Gardent, C. and E. Kow (2007, June). Spotting Overgeneration Suspects. In S. Busemann (Ed.), *Proceedings of the 11th European Workshop on natural language generation (ENLG 2007)*, Number D-07-01 in DFKI Document, Schloss Dagstuhl, Germany, pp. 41–48. DFKI GmbH, Saarbruecken, Germany. ISSN 0946-0098.

Gardent, C. and L. Perez-Beltrachini (2010, August). RTG-based Surface Realisation for TAG. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, Beijing, China, pp. 367–375. Coling 2010 Organizing Committee.

Gécseg, F. and M. Steinby (1997). Tree languages. In G. Rosenberg and A. Salomaa (Eds.), *Handbook of formal languages*, Volume 3: beyond words, pp. 1–68. New York, NY, USA: Springer-Verlag New York, Inc.

Gottesman, B. (2009). Controlled Generation of Input to a Surface Realiser. Master's thesis, European Masters Program in Language and Communication Technologies, Saarland University/Université Nancy 2.

Joshi, A. and Y. Schabes (1997). Tree-adjoining grammars. In G. Rosenberg and A. Salomaa (Eds.), *Handbook of formal languages*, Volume 3: beyond words, pp. 69–123. New York, NY, USA: Springer-Verlag New York, Inc.

Kanazawa, M. (2007, June). Parsing and Generation as Datalog Queries. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic, pp. 176–183. Association for Computational Linguistics, Morristown, NJ, USA.

Koller, A. and J. Hoffmann (2010, May). Waking up a Sleeping Rabbit: On Natural Language Generation with FF. In R. Brafman, H. Geffner, J. Hoffmann, and H. Kautz

(Eds.), *Proceedings of the 20th ICAPS (Short Papers)*, Toronto, Canada, pp. 238–241. The AAAI Press, Menlo Park, California, USA.

Koller, A. and M. Stone (2007, June). Sentence Generation as a Planning Problem. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic, pp. 336–343. Association for Computational Linguistics, Morristown, NJ, USA.

Koller, A. and K. Striegnitz (2002). Generation as Dependency Parsing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA, pp. 17–24. Association for Computational Linguistics, Morristown, NJ, USA.

Kuhlmann, M. (2010). *Dependency Structures and Lexicalized Grammars: An Algebraic Approach*, Volume 6270 of *Lecture Notes in Computer Science*. Germany: Springer-Verlag Berlin Heidelberg.

Maxwell, J. and R. Kaplan (1993, December). The Interface between Phrasal and Functional Constraints. *Computational Linguistics 19*(4), 571–590.

Nederhof, M. (1996). Efficient Generation of Random Sentences. *Journal of Natural Language Engineering 2*(1), 1–13.

Perez-Beltrachini, L. (2009). Using Regular Tree Grammars to Optimise Surface Realisation. Master's thesis, European Masters Program in Language and Communication Technologies, Université Nancy 2/Free University of Bozen-Bolzano.

Pogodalla, S. (2004, May). Computing semantic representations: towards ACG abstract terms as derivation trees. In *Proceedings of Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, Vancouver, British Columbia, Canada, pp. 64–71.

Purdom, P. (1972). A Sentence Generator for Testing Parsers. *BIT Numerical Mathematics 12*(3), 366–375.

Schmitz, S. and J. Le Roux (2008, June). Feature unification in TAG derivation trees. In C. Gardent and A. Sarkar (Eds.), *Proceedings of the 9th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*, Tübingen, Germany, pp. 141–148.

Shieber, S. (2006, April). Unifying Synchronous Tree Adjoining Grammars and Tree Transducers via Bimorphisms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, Trento, Italy, pp. 377–384. Association for Computational Linguistics, Morristown, NJ, USA.

The XTAG Research Group (2001). A Lexicalised Tree Adjoining Grammar for English. Technical report, University of Pennsylvania, Philadelphia, Pennsylvania, USA.

Vijay-Shanker, K. and A. Joshi (1988, August). Feature structures based Tree Adjoining Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics - Volume 2*, Budapest, Hungary, pp. 714–719. Association for Computational Linguistics, Morristown, NJ, USA.

Vijay-Shanker, K., D. Weir, and A. Joshi (1987, July). Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, USA, pp. 104–111. Association for Computational Linguistics, Morristown, NJ, USA.