RTG based surface realisation for TAG

Claire Gardent, Laura Perez-Beltrachini

RTG based surface realisation for TAG

Claire Gardent
CNRS/LORIA
claire.gardent@loria.fr

Laura Perez-Beltrachini
Université Henri Poincaré/LORIA
laura.perez@loria.fr

Abstract

Surface realisation with grammars integrating flat semantics is known to be NP complete. In this paper, we present a new algorithm for surface realisation based on Feature Based Tree Adjoining Grammar (FTAG) which draws on the observation that an FTAG can be translated into a Regular Tree Grammar describing its derivation trees. We carry out an extensive testing of several variants of this algorithm using an automatically produced testsuite and compare the results obtained with those obtained using GenI, another FTAG based surface realiser.

1 Introduction

As shown in (Brew, 1992; Koller and Striegnitz, 2002), Surface Realisation is NP-complete. Various optimisation techniques have therefore been proposed to help improve practical runtimes. For instance, (Kay, 1996) proposes to reduce the number of constituents built during realisation by only considering for combination constituents with non overlapping semantics and compatible indices. (Kay, 1996; Carroll and Oepen, 2005; Gardent and Kow, 2007) propose various techniques to restrict the combinatorics induced by intersective modifiers all applying to the same structure. And (Koller and Striegnitz, 2002; Gardent and Kow, 2007) describe two alternative techniques for reducing the initial search space.

In this paper, we focus on the optimisation mechanisms of two TAG based surface realisers namely, GENI (Gardent and Kow, 2007) and the algorithm we present in this paper namely, RTGEN (Perez-Beltrachini, 2009). GENI's optimisation includes both a filtering process whose aim is to reduce the initial search space and a two step, "substitution before adjunction", tree combination phase whose effect is to delay modifier adjunction thereby reducing the number of intermediate structures being built. In RTGEN on the other hand, the initial FTAG is converted to a Regular Tree Grammar (RTG) describing its derivation trees and an Earley algorithm, including sharing and packing, is used to optimise tree combination.

We compare GENI with several variants of the proposed RTGEN algorithm using an automatically produced testsuite of 2 679 input formulae and relate the RTGEN approach to existing work on surface realisation optimisation.

The paper is structured as follows. We first present the grammar used by both GENI and RTGEN, namely SEMXTAG (Section 2). We then describe the two surface realisation algorithms (Section 3). In Section 4, we describe the empirical evaluation carried out and present the results. Finally, Section 5 situates RTGEN with respect to related work on surface realisation optimisation.

2 SemXTag

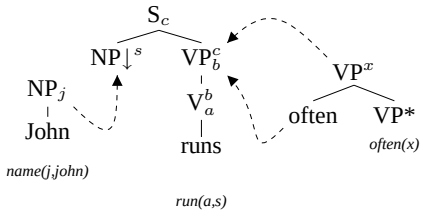
The grammar (SEMXTAG) used by GENI and RTGEN is a Feature-Based Lexicalised Tree Adjoining Grammar (FTAG) augmented with a unification-based semantics as described in (Gardent and Kallmeyer, 2003). We briefly introduce each of these components and describe the grammar coverage. We then show how this FTAG can be converted to an RTG describing its derivation trees.

2.1 FTAG.

A Feature-based TAG (Vijay-Shanker and Joshi, 1988) consists of a set of (auxiliary or initial) elementary trees and of two tree-composition operations: substitution and adjunction. 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. Substitution inserts a tree onto a substitution node of some other tree while adjunction inserts an auxiliary tree into a tree. In an FTAG, 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. Finally, each sentence derivation in an FTAG 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. 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 in (Gardent and Kallmeyer, 2003).



$\Rightarrow$ *name(j,john), run(a,j), often(a)*

Figure 1: Flat Semantics for "John often runs"

Each elementary tree is associated with a flat semantic representation. For instance, in Figure 1,[1] the trees for *John*, *runs* and *often* are associated with the semantics *name(j,john)*, *run(a,s)* and *often(x)* 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 1, the semantic index s occurring in the semantic representation of *runs* 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 s in the tree for *runs* is unified during substitution with the semantic index labelling the root node of the tree for *John*. As a result, the semantics of *John often runs* is {`name(j,john),run(a,j),often(a)`}.

2.3 SemXTAG.

SEMXTAG is an FTAG for English augmented with a unification based compositional semantics of the type described above. Its syntactic coverage approaches that of XTAG, the FTAG developed for English by the XTAG group (The XTAG Research Group, 2001). Like this grammar, 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.

2.4 Converting SemXTAG to RTG

As shown in (Schmitz and Le Roux, 2008), an FTAG can be converted to a Regular Tree Grammar describing its derivation tree. In this section, we briefly sketch this conversion process. For a more precise description of this FTAG to RTG conversion, the reader is referred to (Schmitz and Le Roux, 2008).

[1] C^x/C_x abbreviate a node with category C and a top/bottom feature structure including the feature-value pair $\{$ **index :** $x\}$.

In the FTAG-to-RTG conversion, each SEMXTAG elementary tree is converted to a rule that models its contribution to a TAG derivation tree. A TAG derivation involves the selection of an initial tree, which has some nodes requiring substitution and some permitting adjunction. Let us think of the potential adjunction sites as requiring, rather than permitting, adjunction, but such that the requirement can be satisfied by 'null' adjunction. Inserting another tree into this initial tree satisfies one of the substitution or adjunction requirements, but introduces some new requirements into the resulting tree, in the form of its own substitution nodes and adjunction sites.

Thus, intuitively, the RTG representation of a SEMXTAG elementary tree is a rule that rewrites the satisfied requirement as a local tree whose root is a unique identifier of the tree and whose leaves are the introduced requirements. A requirement of a substitution or adjunction of a tree of root category X is written as X_S or X_A, respectively. Here, for example, is the translation to RTG of the FTAG tree (minus semantics) for *run* in Figure 1, using the word anchoring the tree as its identifier (the upperscripts abbreviates features structures: b/t refers to the bottom/top feature structure and the upper case letters to the semantic index value, $[idx : X]$ is abbreviated to X):

$$S_S^{[t:T]} \rightarrow runs(S_A^{[t:T,b:C]}\ NP_S^{[t:S]}\ VP_A^{[t:C,b:B]}\ V_A^{[t:B,b:A]})$$

The semantics of the SemXTAG tree are carried over as-is to the corresponding RTG rule. Further, the feature structures labelling the nodes of the SemXTAG tree are converted into the RTG rules so as to correctly interact with substitution and adjunction (see (Schmitz and Le Roux, 2008) for more details on this part of the conversion process).

To account for the optionality of adjunction, there are additional rules allowing any adjunction requirement to be rewritten as the symbol ϵ, a terminal symbol of the RTG.

The terminal symbols of the RTG are thus the tree identifiers and the symbol ϵ, and its non-terminals are X_S and X_A for each terminal or non-terminal X of SemXTAG.

3 TAG-based surface realisation

We now present RTGEN and describe GENI, and compare the optimisations they propose to deal with the task complexity.

GENI and RTGEN are similar on several points. They use the same grammar, namely SEMXTAG (cf. Section 2). Further, they both pipeline three main steps. First, **lexical selection** selects from the grammar those elementary trees whose semantics subsumes part of the input semantics. Second, the **tree combining** phase systematically tries to combine trees using substitution and adjunction. Third, the **retrieval phase** extracts the yields of the complete derived trees, thereby producing the generated sentence(s).

GENI and RTGEN differ however with respect to the trees they are working with (derived trees in GENI *vs* derivation trees in RTGEN). They also differ in how tree combination is handled. We now describe these differences in more detail and explain how each approach address the complexity issue.

3.1 GenI

The tree combining phase in GENI falls into two main steps namely, filtering and tree combining.

Filtering. The so-called polarity filtering step aims to reduce the initial search space. It eliminates from the initial search space all those sets of TAG elementary trees which cover the input semantics but cannot possibly lead to a valid derived tree. In specific, this filtering removes all tree sets covering the input semantics such that either the category of a substitution node cannot be canceled out by that of the root node of a different tree; or a root node fails to have a matching substitution site. Importantly, this filtering relies solely on categorial information – feature information is not used. Furthermore, auxiliary trees have no impact on filtering since they provide and require the same category thereby being "polarity neutral elements".

Tree combining. The tree combining algorithm used after filtering has taken place, is a bottom-up tabular algorithm (Kay, 1996) optimised for TAGs. This step, unlike the first, uses all the features

present in the grammar. To handle intersective modifiers, the delayed modifiers insertion strategy from (Carroll et al., 1999) is adapted to TAG as follows. First, all possible derived trees are obtained using only substitution. Next, adjunction is applied. Although the number of intermediate structures generated is still 2^n for n modifiers, this strategy has the effect of blocking these 2^n structures from multiplying out with other structures in the chart.

3.2 RTGen

RTGen synthesises different techniques that have been observed in the past to improve surface realisation runtimes. We first describe these techniques i.e., the main features of RTGEN. We then present three alternative ways of implementing RTGEN which will be compared in the evaluation.

3.2.1 RTGen's main features

A main feature of RTGEN is that it focuses on building derivation rather than derived trees. More specifically, the first two steps of the surface realisation process (lexical selection, tree combining) manipulate RTG rules describing the contribution of the SEMXTAG elementary trees to the derivation tree rather than the elementary tree themselves. The derived trees needed to produce actual sentences are only produced in the last phase i.e., the retrieval phase.

This strategy is inspired from a similar approach described in (Koller and Striegnitz, 2002) which was shown to be competitive with state of the art realisers on a small sample of example input chosen for their inherent complexity. (Koller and Striegnitz, 2002)'s approach combines trees using a constraint based dependency parser rather than an Earley algorithm so that it is difficult to assess how much of the efficiency is due to the parser and how much to the grammar conversion. Intuitively however, the motivation underlying the construction of a derivation rather than a derived tree is that efficiency might be increased because the context free derivation trees (i) are simpler than the mildly context sensitive trees generated by an FTAG and (ii) permit drawing on efficient parsing and surface realisation algorithms designed for such grammars.

Second, RTGEN makes use of the now standard semantic criteria proposed in (Kay, 1996; Carroll et al., 1999) to reduce the number of combinations tried out by the realiser. On the one hand, two constituents are combined by the algorithm's inference rules only if they cover disjoint parts of the input semantics. On the other hand, the semantic indices present in both the input formula and the lexically retrieved RTG trees are used to prevent the generation of intermediate structures that are not compatible with the input semantics. For instance, given the input formula for "John likes Mary", semantic indices will block the generation of "likes John" because this constituent requires that the constituent for "John" fills the patient slot of "likes" whereas the input semantics requires that it fills the agent slot. In addition, chart items in RTGEN are indexed by semantic indices to efficiently select chart items for combination.

Third, RTGEN implements a standard Earley algorithm complete with sharing and packing. Sharing allows for intermediate structures that are common to several derivations to be represented only once – in addition to not being recomputed each time. Packing means that partial derivation trees with identical semantic coverage and similar combinatorics (same number and type of substitution and adjunction requirements) are grouped together and that only one representative of such groups is stored in the chart. In this way, intermediate structures covering the same set of intersective modifiers in a different order are only represented once and the negative impact of intersective modifiers is lessened (cf. (Brew, 1992)). . As (Carroll and Oepen, 2005) have shown, packing and sharing are important factors in improving efficiency. In particular, they show that an algorithm with packing and sharing clearly outtperforms the same algorithm without packing and sharing giving an up to 50 times speed-up for inputs with large numbers of realizations.

3.2.2 Three ways to implement RTGen

Depending on how much linguistic information (i.e. feature constraints from the feature structures) is preserved in the RTG rules, several RTGEN configurations can be tried out which each

reflect a different division of labour between constraint solving and structure building. To experiment with these several configurations, we exploit the fact that the FTAG-to-RTG conversion procedure developed by Sylvain Schmitz permits specifying which features should be preserved by the conversion.

RTGen-all. In this configuration, all the feature structure information present in the SEMXTAG elementary trees is carried over to the RTG rules. As a result, tree combining and constraint solving proceed simultaneously and the generated parse forest contains the derivation trees of all the output sentences.

RTGen-level0. In the RTGen-level0 configuration, only the syntactic category and the semantic features are preserved by the conversion. As a result, the grammar information used by the (derivation) tree building phase is comparable to that used by GENI filtering step. In both cases, the aim is to detect those sets of elementary trees which cover the input semantics and such that all syntactic requirements are satisfied while no syntactic resource is left out. A further step is additionally needed to produce only those trees which can be built from these tree sets when applying the constraints imposed by other features. In GENI, this additional step is carried out by the tree combining phase, in RTGEN, it is realised by the extraction phase i.e., the phase that constructs the derived trees from the derivation trees produced by the tree combining phase.

RTGen-selective. Contrary to parsing, surface realisation only accesses the morphological lexicon last i.e., after sentence trees are built. Because throughout the tree combining phase, lemmas are handled rather than forms, much of the morpho-syntactic feature information which is necessary to block the construction of ill-formed constituents is simply not available. It is therefore meaningful to only include in the tree combining phase those features whose value is available at tree combining time. In a third experiment, we automatically identified those features from the observed feature structure unification failures during runs of the realisation algorithm. We then use only these features (in combination with the semantic features and with categorial information) during tree combining.

4 Evaluation

To evaluate the impact of the different optimisation techniques discussed in the previous section, we use two benchmarks generated automatically from SEMXTAG (Gottesman, 2009).

The first benchmark (MODIFIERS) was designed to test the realisers on cases involving intersective modifiers. It includes 1 789 input formulae with a varying number (from 0 to 4 modifications), type (N and VP modifications) and distribution of intersective modifiers (n modifiers distributed differently over the predicate argument structures). For instance, the formula in (1) involves 2 N and 1 VP modification. Further, it combines lexical ambiguity with modification complexities, i.e. for the *snore* modifier the grammar provides 10 trees.

(1) $l_1 : \exists(x_1, h_r, h_s), h_r \geq l_2, h_s \geq l_3, l_2 : man(x_1), l_2 : snoring(e_1, x_1), l_2 : big(x_1), l_3 : sleep(e_2, x_1), l_4 : soundly(e_2)$
(*A snoring big man sleeps soundly*)

The second benchmark (COMPLEXITY) was designed to test overall performance on cases of differing complexity (input formulae of increasing length, involving verbs with a various number and types of arguments and with a varying number of and types of modifiers). It contains 890 distinct cases. A sample formula extracted from this benchmark is shown in (2), which includes one modification and to different verb types.

(2) $h_1 \geq l_4, l_0 : want(e, h_1), l_1 : \exists(x_1, h_r, h_s), h_r \geq l_1, h_s \geq l_0, l_1 : man(x_1), l_1 : snoring(e_1, x_1), l_3 : \exists(x_2, h_p, h_w, h_u), h_p \geq l_3, h_w \geq l_4, h_u \geq l_5, l_3 : monkey(x_2), l_4 : eat(e_2, x_2, e_3), l_5 : sleep(e_3, x_2)$
(*The snoring man wants the monkey to sleep*)

To evaluate GENI and the various configurations of RTGEN (RTGEN-all, RTGEN-level0, RTGEN-selective), we ran the 4 algorithms in batch mode on the two benchmarks and collected the following data for each test case:

- Packed chart size : the number of chart items built. This feature is only aplicable to RTGen as GENI does not implement packing.

- Unpacked chart size : the number of intermediate and final structures available after unpacking (or at the end of the tree combining process in the case of GENI).
- Initial Search Space (ISS) : the number of all possible combinations of elementary trees to be explored given the result of lexical selection on the input semantics. That is, the product of the number of FTAG elementary trees selected by each literal in the input semantics.
- Generation forest (GF) : the number of derivation trees covering the input semantics.

The graph in Figure 2 shows the differences between the different strategies with respect to the unpacked chart size metric.

A first observation is that RTGEN-all outperforms GENI in terms of intermediate structures built . In other words, the Earley sharing and packing strategy is more effective in reducing the number of constituents built than the filtering and substitution-before-adjunction optimisations used by GENI. In fact, even when no feature information is used at all (RTGEN-level0 plot), for more complex test cases, packing and sharing is more effective in reducing the chart size than filtering and operation ordering.

Another interesting observation is that RTGEN-all and RTGEN-selective have the same impact on chart size (their plots coincide). This is unsurprising since the features used by RTGEN-selective have been selected based on their ability to block constituent combination. The features used in RTGEN-selective mode are `wh`, `xp`, `assign-comp`, `mode`, `definite`, `inv`, `assign-case`, `rel-clause`, `extracted` and `phon`, in addition to the categorial and semantic information. In other words, using all 42 SEMXTAG grammar features has the same impact on search space pruning as using only a small subset of them. As explained in the previous section, this is probably due to the fact that contrary to parsing, surface realisation only accesses the morphological lexicon after tree combining takes place. Another possibility is that the grammar is under constrained and that feature values are missing thereby inducing overgeneration.

Zooming in on cases involving three modifiers,

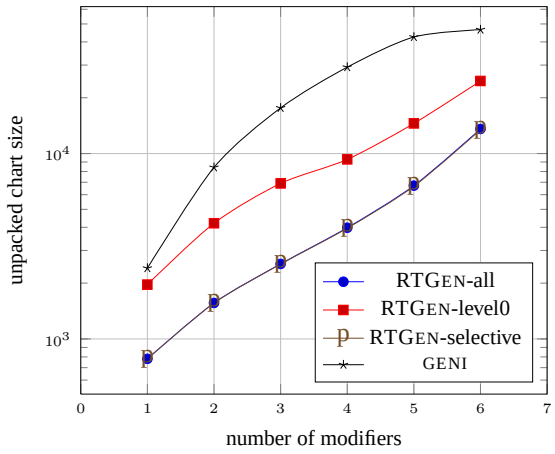


Figure 2: Performance of realisation approaches on the MODIFIERS benchmark, average unpacked chart size as a function of the number of modifiers.

we show in Table 1 the average results for various efficiency metrics [2]. This provides a more detail view of the performance of the differences among the three RTGEN variants.

strategy	GF	chart	unpacked-chart	seconds
RTGen-all	15.05	918.31	2,538.98	0.99
RTGen-level0	1,118.06	2,018	6,898.28	1.41
RTGen-selective	27.08	910.34	2,531.23	0.44

Table 1: Average results on 610 test cases from the MODIFIERS benchmark. Each test case has 3 modifications, distributed in various ways between adjectival and adverbial modifications. 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.

This data shows that running RTGEN with no feature information leads not only to an increased chart size but also to runtimes that are higher in average than for full surface realisation i.e., realisation using the full grammar complete with con-

[2]The two realisers being implemented in different programming languages (RTGEN uses Prolog and GENI Haskell), runtimes comparisons are not necessarily very meaningful. Additionally, GENI does not provide time statistics. After adding this functionality to GENI, we found that overall GENI is faster on simple cases but slower on more complex ones. We are currently working on optimising RTGEN prolog implementation before carrying out a full scale runtime comparison.

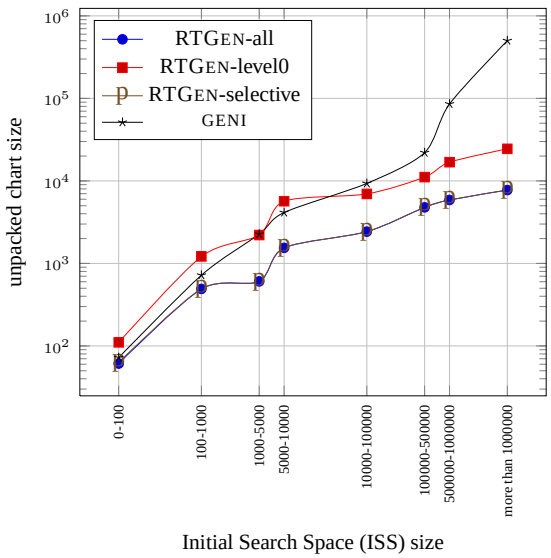


Figure 3: Performance of realisation approaches on the COMPLEXITY benchmark, average unpacked chart size as a function of the ISS complexity.

straints.

Interestingly, it also shows that the selective mode (RTGEN-selective) permits improving runtimes while achieving almost perfect disambiguation in that the average number of derivation trees (GF) produced is close to that produced when using all features. The differences between the two generation forests stems from packing. Using only a subset of features favors packing, thereby reducing the number of chart items built, but introduces over- generation.

Graph 3 and Table 2 confirm the results obtained using the MODIFIERS benchmark on a testset (COMPLEXITY) where input complexity varies not only with respect to modification but also with respect to the length of the input and to the degree of lexical ambiguity. Typically, in a TAG, one word or one semantic literal may be associated either with one tree or with up to several hundred trees (e.g., ditransitive verbs and verbs with several subcategorisation types). By varying the type and the number of verbs selected by the semantic literals contained in the input semantics, the COMPLEXITY benchmark provides a more extensive way to test performance on cases of varying complexity.

strategy	GF	chart	unpacked-chart	seconds
RTGen-all	14.77	693.39	2,427.82	0.81
RTGen-level0	162.02	2,114.16	6,954.84	1.09
RTGen-selective	15.31	692.9	2,427.2	0.36

Table 2: Average results on 335 cases with $10000 < ISS \leq 100000$, from the COMPLEXITY benchmark. The columns show the same performance metrics as in Table 1.

5 Related work

Much work has already been done on optimising surface realisation. Because surface realisation often draws on parsing techniques, work on parsing optimisation is also relevant. In this section, we briefly relate our proposal to another grammar converting approach (Koller and Striegnitz, 2002); to another chart based approach (Carroll and Oepen, 2005); and to approaches based on statistical pruning (White, 2004; Bangalore and Rambow, 2000).

5.1 Optimising surface realisation

Encoding into another grammatical formalism. As already mentioned, the RTGEN approach is closely related to the work of (Koller and Striegnitz, 2002) where the XTAG grammar is converted to a dependency grammar capturing its derivation trees. This conversion enables the use of a constraint based dependency parser, a parser which was specifically developed for the efficient parsing of free word order languages and is shown to support an efficient handling of both lexical and modifier attachment ambiguity.

Our proposal differs from this approach in three main ways. First, contrary to XTAG, SEMXTAG integrates a full-fledged, unification based compositional semantics thereby allowing for a principled coupling between semantic representations and natural language expressions. Second, the grammar conversion and the feature-based RTGs used by RTGEN accurately translates the full range of unification mechanisms employed in FTAG wheras the conversion described by (Koller and Striegnitz, 2002) does not take into account feature structure information. Third, the RTGEN approach was extensively tested on a large benchmark using 3 different configurations whilst (Koller and Striegnitz, 2002) results are re-

stricted to a few hand constructed example inputs.

Chart generation algorithm optimisations. (Carroll and Oepen, 2005) provides an extensive and detailed study of how various techniques used to optimise parsing and surface realisation impact the efficiency of a surface realiser based on a large coverage Head-Driven Phrase Structure grammar.

Because they use different grammars, grammar formalisms and different benchmarks, it is difficult to compare the RTGEN and the HPSG approach. However, one point is put forward by (Carroll and Oepen, 2005) which it would be interesting to integrate in RTGEN(Carroll and Oepen, 2005) show that for packing to be efficient, it is important that equivalence be checked through subsumption, not through equality. RTGEN also implements a packing mechanism with subsumption check, i.e. different ways of covering the same subset of the input semantics are grouped together and represented in the chart by the most general one. One difference however it that RTGEN will pack analyses together as long as the new ones are more specific cases. It will not go backwards to recalculate the packing made so far if a more general item is found (Stefan and John, 2000). In this case the algorithm will pack them under two different groups.

Statistical pruning. Various probabilistic techniques have been proposed in surface realisation to improve e.g., lexical selection, the handling of intersective modifiers or ranking. For instance, (Bangalore and Rambow, 2000) uses a tree model to produce a single most probable lexical selection while in White's system, the best paraphrase is determined on the basis of n-gram scores. Further, to address the fact that there are $n!$ ways to combine any n modifiers with a single constituent, (White, 2004) proposes to use a language model to prune the chart of identical edges representing different modifier permutations, e.g., to choose between *fierce black cat* and *black fierce cat*. Similarly, (Bangalore and Rambow, 2000) assumes a single derivation tree that encodes a word lattice (*a* {*fierce black, black fierce*} *cat*), and uses statistical knowledge to select the best linearisation. Our approach differs from these approaches in that lexical selection is not filtered, intersective modifiers are handled by the grammar (constraints on the respective order of adjectives) and the chart packing strategy (for optimisation), and ranking is not performed. We are currently exploring the use of Optimality Theory for ranking.

6 Conclusion

We presented RTGEN, a novel surface realiser for FTAG grammars which builds on the observation that an FTAG can be translated to a regular tree grammar describing its derivation trees. Using automatically constructed benchmarks, we compared the performance of this realiser with that of GENI, another state of the art realiser for FTAG. We showed that RTGEN outperforms GENI in terms of space i.e. that the Earley sharing and packing strategy is more effective in reducing the number of constituents built than the filtering and substitution-before-adjunction optimisations used by GENI. Moreover, we investigated three ways of interleaving phrase structure and feature structure constraints and showed that, given a naive constraint solving approach, the interleaving approach with selective features seems to provide the best space/runtimes compromise.

Future work will concentrate on further investigating the interplay in surface realisation between phrase structure and feature structure constraints. In particular, (Maxwell and Kaplan, 1994) shows that a more sophisticated approach to constraint solving and to its interaction 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. Further, we intend to integrate Optimality Theory constraints in RTGEN so as support ranking of multiple outputs. Finally, we want to further optimise RTGEN on intersective modifiers using one the methods mentioned in Section 5.

Acknowledgements

The research presented in this paper was partially supported by the European Fund for Regional Development within the framework of the INTERREG IV A Allegro Project.

References

Bangalore, S. and O. Rambow. 2000. Using TAGs, a tree model and a language model for generation. In *Proceedings of TAG+5*, Paris, France.

Brew, Chris. 1992. Letting the cat out of the bag: generation for shake-and-bake mt. In *Proceedings of the 14th conference on Computational linguistics*, pages 610–616, Morristown, NJ, USA. Association for Computational Linguistics.

Carroll, J. and S. Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. *2nd IJCNLP*.

Carroll, J., A. Copestake, D. Flickinger, and V. Paznański. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of EWNLG '99*.

Gardent, C. and L. Kallmeyer. 2003. Semantic construction in FTAG. In *10th EACL*, Budapest, Hungary.

Gardent, C. and E. Kow. 2007. Spotting overgeneration suspect. In *11th European Workshop on Natural Language Generation (ENLG)*.

Gottesman, B. 2009. Generating examples. Master's thesis, Erasmus Mundus Master Language and Communication Technology, Saarbrucken/Nancy.

Kay, Martin. 1996. Chart generation. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 200–204, Morristown, NJ, USA. Association for Computational Linguistics.

Koller, A. and K. Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of the 40th ACL*, Philadelphia.

Maxwell, J. and R. Kaplan. 1994. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4).

Perez-Beltrachini, L. 2009. Using regular tree grammars to reduce the search space in surface realisation. Master's thesis, Erasmus Mundus Master Language and Communication Technology, Nancy/Bolzano.

Schmitz, S. and J. Le Roux. 2008. Feature unification in TAG derivation trees. In Gardent, C. and A. Sarkar, editors, *Proceedings of the 9th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+'08)*, pages 141–148, Tübingen, Germany.

Stefan, Oepen and Carroll John. 2000. Parser engineering and performance profiling. *Journal of Natural Language Engineering*, 6(1):81–98.

The XTAG Research Group. 2001. A lexicalised tree adjoining grammar for english. Technical report, Institute for Research in Cognitive Science, University of Pennsylvannia.

Vijay-Shanker, K. and AK Joshi. 1988. Feature Structures Based Tree Adjoining Grammars. *Proceedings of the 12th conference on Computational linguistics*, 55:v2.

White, M. 2004. Reining in CCG chart realization. In *INLG*, pages 182–191.