Three reasons to adopt TAG-based surface realisation

Claire Gardent
CNRS / LORIA
615, rue du Jardin Botanique
F-54 600 Villers-Lès-Nancy
gardent@loria.fr

Eric Kow
INRIA / LORIA
Université Henri Poincaré
615, rue du Jardin Botanique
F-54 600 Villers-Lès-Nancy
kow@loria.fr

Abstract

Surface realisation from flat semantic formulae is known to be exponential in the length of the input. In this paper, we argue that TAG naturally supports the integration of three main ways of reducing complexity: polarity filtering, delayed adjunction and empty semantic items elimination. We support these claims by presenting some preliminary results of the TAG-based surface realiser GenI.

1 Introduction

Surface realisation consists in producing all the sentences associated by a grammar with a given semantic formula. For lexicalist grammars such as LTAG (Lexicalised Tree Adjoining Grammar), surface realisation usually proceeds bottom-up from a set of flat semantic literals[1]. However, surface realisation from flat semantic formulae is known to be exponential in the length of the input (Kay96; Bre92; KS02). In this paper, we abstract from the TAG based surface realiser for French GenI, (GK05) and argue that TAG naturally supports the integration of various proposals made to help reduce either surface realisation or parsing complexity into a TAG based, lexically driven surface realiser. Specifically, we show:

1. that TAG elementary trees naturally support the implementation of a technique called *polarity filtering* used to reduce the exponential factor introduced by *lexical ambiguity* (Per03),
2. that TAG two operations of substitution and adjunction provides a natural framework for implementing a delayed adjunction mechanism capable of reducing the complexity due to the *lack of ordering information* and
3. that TAG extended domain of locality helps reduce the potential complexity increment introduced by *semantically empty items* such as infinitival *"to"* or complementiser *"that"*.

2 Surface realisation, flat semantics and computational complexity

Why is surface realisation exponential in the length of the input? As shown in (Kay96), one reason for this is the *lack of ordering information*. Contrary to parsing where the input is a string i.e., an ordered list of words, the input to surface realisation is a set of literals. Supposing each literal selects exactly one constituent in the lexicon, then the number of possible combinations between these constituents will be 2^n (the number of subsets obtainable from a set of size n).

In practice of course, there are possible restrictions on constituent combination. In particular, most existing realisers impose the constraint that only constituents with non overlapping semantics and compatible indices can be combined. Because of this restriction, the core of the complexity stems in practice from *intersective modifiers* (Bre92; Kay96). Given a set of n modifiers all modifying the same structure, all possible intermediate structures will be constructed i.e. 2^{n+1}.

A second reason for the exponential complexity of surface realisation is *lexical ambiguity*. As for bottom-up parsing, in surface realisation from flat semantics, the input is used to select a set of lexical entries namely all lexical entries whose seman-

[1]See e.g., (CCFP99) for a discussion summarising the reasons for this choice.

tics subsumes one or more of the input literals. In a realistic grammar, one literal will be associated with more than one lexical entries. So if Lex_i is the number of lexical entries associated with literal l_i, then for an input semantics comprising n literals, the number of sets of lexical constituents covering the input semantics is: $\prod_{i=1}^{i=n} Lex_i$

The two sources of complexity interact by multiplying out so that the potential number of combinations of constituents is:

$$2^n \times \prod_{i=1}^{i=n} Lex_i$$

In what follows, we show that TAG naturally supports various optimisations that have been proposed to reduce the search space.

3 Polarity filtering

To restrict the impact of lexical ambiguity on parsing efficiency, (Per03) introduces a method called *Polarity filtering*. This method is based on the observation that many of the combinations of lexical entries which cover the input semantics are in fact syntactically invalid either because a syntactic requirement is not fulfilled or because a syntactic resource is not used. Accordingly, polarity based filtering eliminates such combinations by:

- assigning each lexical entry with a set of polarities reflecting its syntactic requirements and resources,
- computing for each possible combination of lexical entries the sum of its polarities and
- only allowing surface realisation on combinations which have a net sum of zero (all requirements are satisfied and all resources are used).

By filtering the initial search space before the tree combination phase, polarity filtering in effect reduces the impact of lexical ambiguity i.e. decreases $\prod_{i=1}^{i=n} Lex_i$.

The definitory properties of TAG elementary trees provide a natural way to assign polarities to a TAG lexical entries: each elementary tree can be associated with a polarity $+C$, where C is the category of its root node and each substitution or foot node in that tree, a polarity $-C$ is added, where C is the category of that node.

We implemented polarity filtering in GenI based on this way of associating lexical entries with polarities[2]. We then measured the impact of this filtering on the initial search space (the number of sets of lexical items actually explored by the realiser), on space (measured by the number of chart items created) and on time.

Table 1 summarises the impact of polarity filtering on the initial search space[3]. **possible** indicates the number of combinations of lexical entries which cover the input semantics and thus can potentially lead to a valid syntactic tree realising the input semantics and **explored** gives the number of combinations actually explored by the surface realiser after polarity filtering has ruled out combinations which cannot possibly lead to a valid syntactic tree).

As is to be expected, the impact increases with the number of input literals so that while polarity filtering divides the initial search space by 35.6 for an input ranging between 1 and 6 literals, it divides it by 441.6 for an input size ranging between 14 and 16 literals

literals	**possible**	**explored**	**($\times$)**
1-6	199.10	5.60	35.6
7-9	6460.88	40.06	161.3
10-13	43028.25	137.06	313.9
14-16	292747.64	662.91	441.6

Figure 1: Polarity filtering and initial space
(Sets of initial trees covering the input semantics)

Table 2 gives the impact of polarity filtering on space as measured by the number of created chart items (or constituents). The first column (**w/o pol.**) gives the number of created charted items when polarity filtering is switched off and the second, (**with pol.**) when polarity filtering is on. As can be seen, the effect is particularly pronounced when the input exceeds 10 literals.

Finally, Figure 3 shows that the overhead introduced by the construction of the polarity automaton means that formulae under 10 literals are realised in roughly the same time with or without polarity filtering. However, for larger sentences, polarity filtering is increasingly important in keeping realisation times reasonable. For instance, given an input ranging between 14 and 16 literals, polar-

[2]See (GK05) for more details.

[3]For each group of input (1-6 literals, 7-9, etc.), measures are based on an average of 15 cases.

literals	w/o pol.	with pol.	(×)
1-6	146.40	83.60	1.8
7-9	3273.50	1281.25	2.6
10-13	7468.06	702.50	10.6
14-16	17502.36	1613.91	10.8

Figure 2: With and without Polarity filtering (Chart items)

ity filtering divides realisation time by 5, that is, yields a realisation time of 2.21 seconds instead of 11.61.

literals	w/o pol.	with pol.	(×)
1-6	0.81	0.79	1.0
7-9	1.68	1.35	1.2
10-13	3.56	1.88	1.9
14-16	11.61	2.21	5.3

Figure 3: With and without Polarity filtering (CPU times)

4 Substitution/adjunction distinction

One important specificity of TAG is that it includes two combination operations namely, adjunction and substitution. We now show that this feature of TAG is particularly useful in improving surface realisation performance.

4.1 Reducing the impact of intersective modifiers

To restrict the combinatorics induced by modifiers, (CCFP99; CO05) proposes either to handle modifiers after a complete syntactic tree is built (i.e., after all syntactic requirements are fulfilled) or before the modifiee is combined with other items (e.g., before the head noun has combined with a determiner). Although the number of intermediate structures generated is still 2^n for n modifiers, both strategies have the effect of blocking these 2^n structures from multiplying out with other structures in the chart. More precisely, given an input semantics of size n where k of its literals are to be realised as modifiers, the number of intermediate structures possible in the two phase approach is $2^k + 2^{n-k}$, which can be considerably smaller than 2^n, depending on the size of k.

In TAG, we can make use of the fact that substitution and adjunction apply independently of each other to implement a two-phase generation strategy where modifiers are handled only after a complete syntactic tree is built. In the first phase, only substitutions are performed and in the second, only adjunctions. Additionally, before adjunction starts, all unsaturated trees (trees with unfilled substitution sites) are discarded from the chart thereby ensuring that modifiers do not combine with structures that cannot possibly lead to a valid result (since no constituent could be found to fill the unsaturated substitution sites).

Since in TAG, modifiers always involve the use of adjunction, modifiers will always be handled by the second phase of the algorithm and thereby adjoined into "saturated trees" i.e., trees devoid of unfilled substitutions sites. In this way, the proliferation of structures induced by the modifiers can be restricted.

The substitution-before-adjunction strategy was integrated in `GenI` yielding the improvements indicated in Figures 4 and 5.

literals	1 phase	2 phase	(×)
≤ 3	0.73	0.73	1.0
4	0.74	0.75	1.0
5	0.97	0.93	1.0
6	2.91	0.89	3.3
7	4.24	1.30	3.3
≥ 8	Time out		

Figure 4: With and without SBA (CPU times)

literals	1 phase	2 phase	(×)
≤ 3	47.00	44.33	1.1
4	107.00	108.00	1.0
5	310.00	263.00	1.2
6	1387.33	883.00	1.6
7	2293.50	761.33	3.0

Figure 5: With and without SBA (Chart items)

As table 4 shows, when there is more than 7 literals in the input, the one-phase algorithm times out. More in general, for the data shown, the two phase strategy leads to an average decrease in time ranging between 1 and 3.3% and a decrease in space varying between 1.1% and 3% respectively.

Although the poor performance of the 1 phase algorithm is in part due to a very large and strongly overgenerating grammar[4] , the data clearly shows that SBA is essential in supporting large scale TAG based surface realisation.

[4]The grammar used is a grammar for French which contains roughly 3 400 initial trees (CD04).

4.2 Substitution-before-adjunction combined with Polarity Filtering

The substitution-before-adjunction strategy limits the impact of intersective modifiers by restricting the number of constituents the modifiers can combine with *within one set of lexical items*. Because polarity filtering reduces the number of sets of lexical items to be considered, it trivially also reduces the number of sets of lexical items involving adjunctions.

The space improvement provided by combining the substitution-before-adjunction (SBA) strategy with polarity filtering is illustrated in Figures 6 and 7 which show the space reduction associated with cases ordered either according to their number of literals or according to their number of foot nodes (i.e., adjunction cases). As should be expected, the number of foot nodes is more highly correlated with a space reduction. Specifically, a combined SBA/polarity strategy divides by 3.4 the space used for cases involving between 1 and 12 auxiliary trees; and by 18.8 the space used for cases involving between 14 and 16 auxiliary trees.

literals	**w/o pol.**	**with pol.**	**(×)**
1-6	367.90	109.50	3.4
7-9	6192.69	1550.19	4.0
10-13	11211.06	711.06	15.8
14-16	30660.27	1631.64	18.8

Figure 6: SBA + Polarity (Chart items)

# aux trees	**w/o pol.**	**with pol.**	**(×)**
1-12	2124.27	620.82	3.4
13-120	8751.53	1786.47	4.9
121-190	11528.43	611.50	18.9
191-350	25279.75	1085.75	23.3

Figure 7: SBA + Polarity (Chart items)

4.3 Filtering out unusable trees

Another interesting aspect of TAG's use of two combination operations and more specifically of the substitution-before-adjunction strategy is that it naturally supports the inclusion of a third phase to filter out unusable trees that is, trees which can be determined not to be integrable in any valid derivation. Specifically, this third phase occurs between substitution and adjunction and filters out:

- all trees with an unfilled substitution site
- all saturated trees whose root node is not labelled with an S category

The first filter (elimination of unsaturated trees) is required, as indicated above, to restrict the impact of intersective modifiers: by discarding them, we restrict adjunction to saturated trees. The second, makes use of the property of auxiliary trees which insists that root and foot node be labelled with the same category. Because of this property, adjunction cannot affect the category of the tree it adjoins to. In particular, a tree which after all possible substitutions have been performed, has root label C with $C \neq S$ can never lead to the creation by adjunction of a tree with root label S. Hence it can be discarded (provided of course, the generator is seeking to build sentences).

Figures 8 and 9 illustrate the impact of this second filter (called the *Root Node Filter*, RNF) on the chart size when polarity filtering is switched off. As for SAB, the figures show a higher correlation between the RNF and the number of adjunction nodes than with the number of literals. Intriguingly, the impact of the filter is proportionally higher on sentences with fewer foot nodes. Although this needs to be checked more thoroughly, the explanation for this could be the following. The trees removed by the Root Node Filter are saturated tree not rooted in S hence essentially saturated NP trees. Examination of the data reveals that the number of these trees removed by the RNF remains almost constant (though this might be an ad hoc property of the specific testsuite used). Hence in proportion, the effect of the RNF diminishes.

Note however that in absolute terms, the number of trees whose derivation is avoided by the RNF remains quite high thus contributing to an overall better performance.

literals	**w/o RNF**	**with RNF**	**(×)**
1-6	367.90	146.40	2.5
7-9	6192.69	3273.50	1.9
10-13	11211.06	7468.06	1.5
14-16	30660.27	17502.36	1.8

Figure 8: Root node filter w/o Pol (Chart Items).

As Figures 10 and 11 show, combining the Root Node Filter with polarity filtering simply reinforces the biases noted above: Root Node Filtering is proportionally more effective for short input but can remain useful in absolute terms. A more thor-

# aux trees	w/o RNF	with RNF	(×)
1-12	2124.27	527.36	4.0
13-120	8751.53	5570.33	1.6
121-190	11528.43	6490.14	1.8
191-350	25279.75	15469.17	1.6

Figure 9: Root node filter w/o Pol (Chart Items).

ough investigation of the data and further experiments are needed however to determine whether such behaviour is not tied to some ad hoc property of our (still too limited) testsuite.

literals	w/o RNF	with RNF	(×)
1-6	109.50	83.60	1.3
7-9	1550.19	1281.25	1.2
10-13	711.06	702.50	1.0
14-16	1631.64	1613.91	1.0

Figure 10: Root node filter + Pol (Chart Items).

# aux trees	w/o RNF	with RNF	(×)
1-12	422	621	1.5
13-120	1627	1786	1.1
121-190	600	612	1.0
191-350	1073	1086	1.0

Figure 11: Root Node Filter + Pol (Chart Items).

5 TAG extended domain of locality

Arguably there are words such as complementiser *that* or infinitival *to* whose semantics is empty. These words are to surface realisation what gaps (or empty categories) are to parsing. In a naive approach, they require that all trees with an empty semantics be considered as potential constituent candidate at each combining step. In terms of efficiency, this roughly means increasing the size of the input n (just like postulating gaps at all position in an input string increases the size of that string).

To avoid this shortcoming, a common practice (CCFP99) consists in specifying a set of rules which selects empty semantic items on the basis of the input literals. However these rules fail to reflect the fact that empty semantic items are usually functional words and hence governed by syntactic rather than semantic constraints.

By contrast, in a TAG based surface realiser, TAG elementary trees provide a natural way to specify the syntactic environment in which empty semantic items can occur. For instance, complementiser *that* occurs with verbs taking a sentential argument which is generally captured by including the complementiser as a co-anchor in the trees of these verbs.

More in general, the extended domain of locality provided by TAG elementary trees, together with the possibility of specifying co-anchors means that empty semantic items can be avoided altogether. Hence they do not require specific treatment and have no impact on efficiency.

6 Discussion

We have argued that TAG presents several features that makes it particularly amenable to the development of an optimised surface realiser. We now summarise these features and briefly compare TAG with CCG (Combinatory Categorial Grammar) and HPSG (Head Driven Phrase Structure Grammar) based surface realisation.

6.1 Using tree node types

The *different types of tree nodes* identified by TAG can be used to support polarity filtering whereby substitution nodes can be associated with negative polarities (requirements) and root nodes with positive polarities (resources). As our preliminary experiments show, polarity filtering has a significant impact on the initial search space, on the space used and on CPU times.

So far, this particular type of global filtering on the initial search space has been used neither in the HPSG (CCFP99; CO05) nor in the CCG (Whi04) approach. Although it could presumably be adapted to fit these grammars, such an adaptation is in essence less straightforward than in TAG.

In CCG, the several combination rules mean that a subcategory can function either as a resource or as a requirement depending on the rule that applies. For instance, in the verbal category $(S\backslash NP)/NP$, the subcategory $S\backslash NP$ functions as a resource when NPs are type raised (it satisfies the requirement of a type raised NP with category $S/(S\backslash NP)$). However it will need to be further decomposed into a resource and a requirement if they are not. More in general, polarity specification in CCG would need to take into account the several combination rules in addition to the category structure. In HPSG, it is the interaction of lexical categories with lexical and phrasal rules that will need to be taken into consideration.

6.2 Using rule types

The *two types of tree combining operations* permitted by TAG can be used to structure the surface realisation algorithm. As we've shown, performing all substitutions before allowing for adjunction greatly reduces the exponential impact of intersective modifiers. Moreover, combining such a substitution-before-adjunction strategy with polarity filtering further improves performance.

In comparison, the HPSG and the CCG approach do not support such a natural structuring of the algorithm and intersective modifiers induce either a pre- or a post-processing.

In HPSG, intersective modifiers are discarded during the chart generation phase and adjoined into the generated structures at a later stage. This is inelegant in that (i) intersective modifiers are artificially treated separately and (ii) structures subject to adjunction have to be non monotonically recomputed to reflect the impact of the adjunction in that part of the tree dominating the adjunction.

In CCG, the input logical form is chunked into subtrees each corresponding to a separate generation subproblem to be solved independently. Again the approach is ad hoc in that it does not rely on a given grammatical or linguistic property. As a result, e.g., negation needs special treatment to avoid incompleteness (if the heuristic applies, negated sentences cannot be generated). Similarly, it is unclear how long distance dependencies involving modifiers (e.g., *Which office did you say that Peter work in ?*) are handled.

6.3 Using TAG extended domain of locality

TAG extended domain of locality means that empty semantic items need no special treatment. In contrast, both the HPSG and the CCG approach resort to ad hoc filtering rules which, based on a scan of the input semantics, add semantically empty items to the chart.

7 Further research

Although the results presented give strong evidence for the claim that TAG naturally supports the development of an optimised surface based realiser, they are based on a limited testsuite and on a core grammar for French that heavily overgenerates. Hence they do not truly reflect the potential of the proposed optimisations on the performance of a large scale surface realiser. Current work concentrates on remedying these shortcomings. In particular, we are working on developing a *structured* test suite which permits a precise measure of the impact of different factors both on complexity and on the optimisations used. In this testsuite for instance, each item is associated with a series of indicators concerning its potential complexity: number of literals in the corresponding input semantics, number of trees, number of nodes, number of substitutions nodes and number of foot nodes in the corresponding selection of initial trees.

Further work also includes restricting overgeneration and exploring in how far, polarity filtering can be used to select one among the many paraphrases

References

C. Brew. Letting the cat out of the bag: Generation for shake-and-bake MT. In *Proceedings of COLING '92*, Nantes, France, 1992.

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

B. Crabbé and D. Duchier. Metagrammar redux. In *International Workshop on Constraint Solving and Language Processing - CSLP 2004, Copenhagen*, 2004.

J. Carroll and S. Oepen. High efficiency realization for a wide-coverage unification grammar. In R. Dale and K-F. Wong, editors, *Proceedings of the Second International Joint Conference on Natural Language Processing*, volume 3651 of *Springer Lecture Notes in Artificial Intelligence*, pages 165–176, 2005.

C. Gardent and E. Kow. Generating and selecting grammatical paraphrases. In *Proceedings of the 10th European Workshop on Natural Language Generation*, Aberdeen, Scotland, 2005.

M. Kay. Chart Generation. In *34th ACL*, pages 200–204, Santa Cruz, California, 1996.

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

G. Perrier. Les grammaires d'interaction, 2003. Habilitation à diriger les recherches en informatique, université Nancy 2.

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