Spotting Overgeneration Suspects

Claire Gardent
CNRS/LORIA
Nancy, France
claire.gardent@loria.fr

Eric Kow
INRIA/LORIA/UHP
Nancy, France
eric.kow@loria.fr

Abstract

We present a method for quickly spotting overgeneration suspects (i.e., likely cause of overgeneration) in hand-coded grammars. The method is applied to a medium size Tree Adjoining Grammar (TAG) for French and is shown to help reduce the number of outputs by 70% almost all of it being overgeneration.

1 Introduction

A generative grammar should describe all and only the sentences of the language it describes. In practice however, most grammars both under- and overgenerate. They under-generate in that they fail to describe all the language sentences and they overgenerate in that they licence as grammatical, strings that are not.

In a computational setting, this theoretical shortcoming means that processing will yield either too many or too few sentence analyses. Undergeneration results in insufficient coverage (some sentences cannot be parsed). Conversely, overgeneration leads to an explosion of generated strings.

Here we focus on overgeneration. There are several reasons why grammars overgenerate.

First, as is now well-known, grammar engineering is a highly complex task. It is in particular, easy to omit or mistype a constraint thereby allowing for an illicit combination and indirectly, an illicit string.

Second, a computational grammar is a large object and predicting all the interactions described by even a medium size grammar is difficult, if not impossible. Indeed this is why a surface realiser that produces all the strings associated with a given semantics is a valuable tool: it permits checking these predictions on concrete cases.

Third, grammars are often compiled from more abstract specifications and this additional layer of abstraction introduces the risk of licensing an illicit elementary structure. This is the case in particular in our approach where the TAG used by the realiser is compiled from a so-called "metagrammar" (cf. section 2). As we shall see in section 4, this added level of abstraction means that elementary trees are present in the grammar that shouldn't. These trees may also induce overgeneration.

In this paper, we propose a method for reducing overgeneration in a computational grammar. We apply the proposed approach to a Tree Adjoining Grammar for French (SEMFRAG) and show that it results in a 70% decrease of the generation output on a graduated test suite of 140 input semantics.

The paper is structured as follows. We start (section 2) by presenting the computational framework in which our experiment is based namely SEMFRAG, a tree adjoining grammar for French and GENI, a surface realiser. In section 3, we then go on to describe the methodology we propose to identify and eradicate sources of overgeneration. Section 4 presents the results of the evaluation. Section 5 concludes with pointers for further research.

2 The computational framework

We now briefly describes the GENI surface realiser and the SEMFRAG TAG on which we tested our debugging method.

2.1 SemFraG, a TAG for French integrating semantic information

SEMFRAG is a Feature-based lexicalised TAG (FTAG, (VSJ88)) for French extended with semantic information as described in (GK03).

A Feature-based TAG (FTAG, (VSJ88)) 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 labelled 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.

To associate semantic representations with natural language expressions, the FTAG is modified as proposed in (GK03).

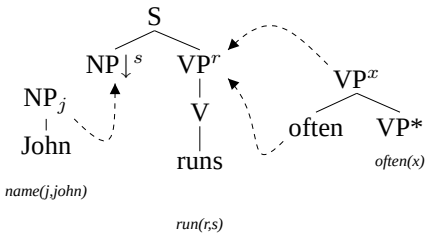

Figure 1: Flat semantics for "John often runs"

Each elementary tree is associated with a flat semantic representation[1]. For instance, in Figure 1, the trees[2] for *John, runs* and *often* are associated with the semantics *name(j,john)*, *run(r,s)* and *often(x)* respectively.

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. In the same example, 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 indices labelling the root nodes of the tree for *John*. As a result, the semantics of *John often runs* is

(1) {name(j,john),run(r,j),often(r)}

The grammar used describes a core fragment for French and contains around 6 000 elementary trees. It covers some 35 basic subcategorisation frames and for each of these frames, the set of argument redistributions (active, passive, middle, neuter, reflexivisation, impersonal, passive impersonal) and of argument realisations (cliticisation, extraction, omission, permutations, etc.) possible for this frame. As a result, it captures most grammatical paraphrases that is, paraphrases due to diverging argument realisations or to different meaning preserving alternation (e.g., active/passive or clefted/non clefted sentence).

2.2 SemFraG and XMG

SEMFRAG is compiled from a higher-level XMG (eXtensible MetaGrammar) specification (CD04). Briefly, the XMG formalism permits specifying basic classes and then combining them (by inheritance, conjunction and disjunction) to produce SEMFRAG elementary trees and their associated semantics (cf. (CD04; Gar06)). For instance, the tree for an active intransitive verb taking a nominal canonical subject will result from specifying and then conjoining classes for the canonical nominal subject, the active verb form and the unary relation.

Importantly, the compilation process keeps track of which classes are used to produce each elementary tree. As a result, each SEMFRAG elementary tree is associated with the set of classes used to produce that tree. For instance, in SEMFRAG, the tree for the active form of intransitive verbs taking a nominal canonical subject will be associated by the XMG compiler with the following set of properties:

CanonicalSubject, n0Vn1, ActiveForm, UnaryRel, NonInvertedNominalSubject

More generally, the set of classes associated by the XMG compilation process with each elementary

[1]The examples given actually show a simplified version of the flat semantics used y GENI where in particular, so-called labels are omitted. A full specification is given in (**?**).

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

tree (we will call this the *tree properties*) provides clear linguistic information about these trees. As we shall see in section 3, this information is extremely useful when seeking to identify *overgeneration suspects* i.e., elementary trees which are likely to cause overgeneration.

2.3 The GenI surface realiser

The basic surface realisation algorithm[3] used is a bottom up, tabular realisation algorithm (Kay96) optimised for TAGs. It follows a three step strategy which can be summarised as follows. Given an empty agenda, an empty chart and an input semantics ϕ:

Lexical selection. Select all elementary trees whose semantics subsumes (part of) ϕ. Store these trees in the agenda. Auxiliary trees devoid of substitution nodes are stored in a separate agenda called the auxiliary agenda.

Substitution phase. Retrieve a tree from the agenda, add it to the chart and try to combine it by substitution with trees present in the chart. Add any resulting derived tree to the agenda. Stop when the agenda is empty.

Adjunction phase. Move the chart trees to the agenda and the auxiliary agenda trees to the chart. Retrieve a tree from the agenda, add it to the chart and try to combine it by adjunction with trees present in the chart. Add any resulting derived tree to the agenda. Stop when the agenda is empty.

When processing stops, the yield of any syntactically complete tree whose semantics is ϕ yields an output i.e., a sentence.

The workings of this algorithm can be illustrated by the following example. Suppose that the input semantics is (1). In a first step (**lexical selection**), the elementary trees selected are the ones for *John, runs, often*. Their semantics subsumes part of the input semantics. The trees for *John* and *runs* are placed on the agenda, the one for *often* is placed on the auxiliary agenda.

The second step (the **substitution phase**) consists in systematically exploring the possibility of combining two trees by substitution. Here, the tree for *John* is substituted into the one for *runs*, and the resulting derived tree for *John runs* is placed on the agenda. Trees on the agenda are processed one by one in this fashion. When the agenda is empty, indicating that all combinations have been tried, we prepare for the next phase.

All items containing an empty substitution node are erased from the chart (here, the tree anchored by *runs*). The agenda is then reinitialised to the content of the chart and the chart to the content of the auxiliary agenda (here *often*). The **adjunction phase** proceeds much like the previous phase, except that now all possible adjunctions are performed. When the agenda is empty once more, the items in the chart whose semantics matches the input semantics are selected, and their strings printed out, yielding in this case the sentence *John often runs*.

3 Reducing overgeneration

We now present the methodology we developed for identifying and eradicating sources of overgeneration. In essence, the idea is to first, manually annotate the realiser output as either PASS or OVERGENERATION and to then use the annotated data to:

> *automatically spot the items ((sets of) TAG elementary trees, pairs of combined trees) which systematically occur only in overgeneration cases.*

More specifically, the procedure we defined to reduce overgeneration can be sketched as follows (cf. also Figure 2).

1. Surface realisation is applied to a graduated test suite of input semantics thus producing a (detailed) derivation log of all the derivations associated with each input in the testsuite
2. The outputs given by the derivation log are (manually) classified into PASS or OVERGENERATION sentences, the overgeneration mark indicating strings that either do not actually belong in the target language, or should not be associated to the input semantics.
3. The annotated output is used to automatically produced a *suspects report* which identifies a list of suspects i.e., a list of TAG trees or derivation steps which are likely to cause the overgeneration because they only occur in overgeneration cases.
4. The grammar is debugged and re-executed on the data

[3] See (GK05) for more details.

5. The derivations results are compared with the previous ones and any discrepancy (less or more sentences generated) signalled.

In a sense, this is an approach that might already be widespread in generation: produce some output, and correct the grammar possibly with the aid of a derivation log. Our contributions are a systematic, incremental approach; a high level of automation, which increases our throughput by focusing human attention on correcting the grammar rather than the unrelated details; and research into summarisation of the operations log so that we can more easily identify the source of error.

3.1 An incremental approach

First experiments with SEMFRAG showed that the grammar strongly overgenerates both because it was initially developed for parsing and because it is automatically compiled from an abstract specification (cf. section 1). Indeed for some inputs, the realiser produced over 4000 paraphrases, a large portion of them being overgeneration. More generally, Figure 3 shows that the number of outputs for a given input varies between 0 and 4908 with an average of 201 outputs per input (the median being 25).

To avoid having to manually annotate large amounts of data, we relied on a graduated test suite and proceeded through the data from simplest to more complex. Concretely, this means that we first eliminated overgeneration in input corresponding to sentences with one finite verb (INPUT1) before moving on to inputs corresponding to sentences with two (INPUT2) and three (INPUT3) finite verbs. This means that as we moved from the simplest to the more complex data, overgeneration was incrementally reduced thereby diminishing the number of output to be annotated.

Indeed this worked very well as by simply looking at INPUT1 we achieved a 70% decrease in the number of outputs for the total testsuite (cf. section 4).

3.2 Semi-automated grammar debugging

The debugging procedure described above was implemented through a test harness interleaving manual annotations with machine-generated output. Three points are worth stressing. First, the suspects report is produced automatically from the annotated derivation log. That is, except for the derivation log manual annotation, the identification of the suspects information is fully automated. Second, regression testing is used to verify that corrections made to the grammar do not affect its coverage (all PASS remains PASS). Third, the harness provides a linguist friendly environment for visualising, modifying and running the grammar on the inputs being examined.

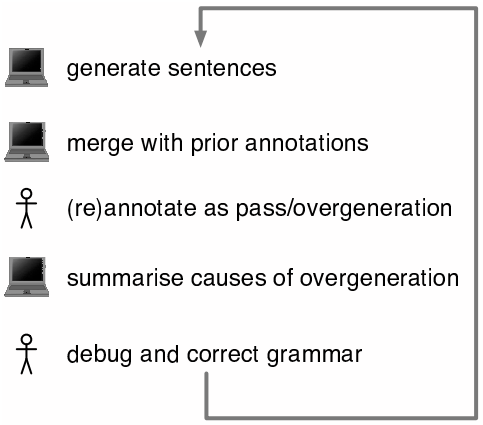

Figure 2: Test harness

3.3 Listing the suspects

The derivation log produced by GENI contains detailed information about each of the derivations associated with a given input. More specifically, for each generated string, the derivation log will show the associated derivation tree together with the tree family, tree identifier and tree properties associated with each elementary tree composing that derivation tree.

```
Output: jean se demande si c'est
        paul qui vient
demander:n8 <-(s)- venir
demander:n1 <-(s)- jean
venir:n4 <-(s)- paul

demander Tn0ClVs1int-630
  CanonicalSubject
  NonInvertedNominalSubject
  SententialInterrogative
venir Tn0V-615
  CleftSubject
  NonInvertedNominalSubject
paul TproperName-45
jean TproperName-45
```

However, the derivation log can be both very long and very redundant. To extract from it information that points more directly to the likely causes of overgeneration, we first manually annotate each string

as *pass* or *overgeneration*. We then automatically extract from the annotated derivation log, a much shorter "suspects report" which identifies suspects i.e., likely causes for overgeneration.

In essence, this suspects report lists trees, sets of trees or derivation items that *only occur in overgeneration cases* (i.e., strings that have manually been annotated as OVERGENERATION). Moreover, information about the suspects is displayed in a compact and informative way. Specifically, for a given generation input, the suspects report will consist of a (possibly empty) list of items of the following form[4]:

1. Lemma
2. TreeFamily ?(all) – ?($\wedge$ tree-property)
3. ?(TreeId* ? ($\wedge$ tree-property))
4. ?(TreeId$_i$:nodeId$_j$ $\xleftarrow{Op}$ TreeId$_k$)

That is, a suspect report item (SR-ITEM) indicates, for a given lemma (line 1), the tree family (line 2), the specific trees (line 3), the specific tree properties (lines 2 and 3) and/or the specific derivation items[5] (line 4) that consistently occur only in overgeneration cases.

The suspects report is compact in that it only outputs information about likely suspects i.e., trees, tree family, tree properties and/or derivation items which consistently occur only in overgeneration cases. Furthermore, it groups together overgeneration sources which share a common feature (same tree family, same tree family and same tree properties, same derivation items). As we shall see, displaying the commonalities between suspects makes it easier for the linguist to understand the likely cause of overgeneration (for instance, if all the trees of a given family lead to overgeneration, then it is likely that the grammar is not sufficiently constrained to block the use of this family in the particular context considered).

It is informative in that it gives detailed information about the likely cause of overgeneration. In particular, tree properties can be extremely useful in understanding the commonalities between the trees involved and thereby the likely cause of overgeneration.

[4]The $*$ is the Kleene star, ? indicates optionality.

[5]A derivation item of the form TreeId$_i$:nodeId$_j$ $\xleftarrow{Op}$ TreeId$_k$:nodeId$_l$ indicates that TreeId$_k$ has been added to the node nodeId$_j$ of TreeId$_i$ using the operation Op where Op is either adjunction or substitution.

To better illustrate the type of information contained in a suspects report, we now go through a few examples.

Example 1: "il faut partir/? devoir partir" Given the input semantics for e.g., *il faut partir* (we must go), the suspects report tells us that the presence in a derivation of any trees of the family `SemiAux` leads to overgeneration.

```
consistent overgeneration for devoir
TSemiAux (all) - SemiAux
  [506]
```

Indeed, in this context (i.e., given the input semantics considered), the use of a `SemiAux` tree results in the production of such strings as *devoir partir* which are grammatical but do not yield a finite sentence as output. If desired, this particular overgeneration bug can be fixed by constraining the generator output to be a finite sentence.

Example 2: "Jean dit accepter/*C'est par Jean qui accepte qu'être dit". In the previous example, the SR-ITEM indicates that all trees of a given family lead to overgeneration but there is only one tree in that family. A more interesting case is when there are several such trees. For instance, the SR-ITEM below indicates that all derivations involving an n0Vn1 tree anchored with *dire* lead to overgeneration and that there are 6 such trees (trees 699 ... 750). Moreover the tree properties information indicates that all these trees share the `InfinitiveSubject Passive` tree properties. Inspection of the data shows that these trees combine with a finite form of *accepter* to yield highly agrammatical strings such as *c'est par Jean qui accepte qu'être dire* (instead of e.g., *Jean dit accepter*. In short, the SR-ITEM indicates that the grammar is not sufficiently constrained to block the combination of the infinitive passive form of the n0Vn1 trees anchored with *dire* with some of the trees associated by the grammar with *accepter*.

```
input t90
Lemma: dire
Tn0Vn1 (all) - InfinitiveSubject
        Passive
  [699] CanonicalCAgent Passive
  [746] CanonicalGenitive dePassive
  [702] CleftCAgentOne Passive
  [752] CleftDont dePassive
  [751] CleftGenitiveOne dePassive
  [750] RelativeGenitive dePassive
```

Example 3: "Jean doit partir/*C'est Jean il faut que qui part" Sometimes overgeneration will only occur with some of a family trees and in this case the third line of the SR-ITEM indicates which are those trees and which are their distinguishing properties (i.e. the properties that always result in overgeneration). For instance, the suspects report for the input semantics of *Jean doit partir* (Jean must leave), contains the following single SR-ITEM:

```
Input t30
consistent overgeneration for partir
Tn0V - CleftSubject
  [604]
```

This indicates that all derivations including tree 604 of the n0V family anchored with *partir* lead to overgeneration. Indeed such derivations license highly ungrammatical sentences such as *C'est Jean il faut que qui part* where a cleft subject tree for *partir* combines with the canonical tree for *il faut*. This overgeneration bug can be fixed by constraining n0V cleft subject trees to block such illicit combinations.

Example 4: "L'homme riche part/* riche l'homme part" Finally, overgeneration may sometimes be traced back to a specific derivation item i.e., to a specific tree combination. This will then be indicated in the last line of the trace item. For instance, the following SR-ITEM indicates that adjoining the adjective auxiliary tree `Tn0vA-90` to the root of a determiner tree always lead to overgeneration. Indeed such an adjunction results in sentences where the adjective precedes the determiner which in French is agrammatical.

```
Input t70
consistently overgenerating derivation
item
le:Tdet-17:n0 <-(a)- riche:Tn0vA-90
```

4 Results and Evaluation

4.1 Before and after figures

We have used the test harness over a period of one week, roughly 12 consecutive man hours. Over that period we have run over ten iterations of the test harness, making 13 modifications to the grammar as a result. In the process of revising this grammar, we have studied 40 cases (under one third of the whole suite) and manually annotated 1389 outputs with pass/overgeneration judgements. On the whole 140 cases of the test suite, the original grammar produced 28 167 outputs (4908 for the worst case, 201 mean, 25 median). The revised grammar produces 70% fewer likely agramatical outputs, leaving behind 8434 sentences (201 worst case, 60 mean, 12 median). We believe that this reduction is especially noteworthy given the little time we have spent in this process.

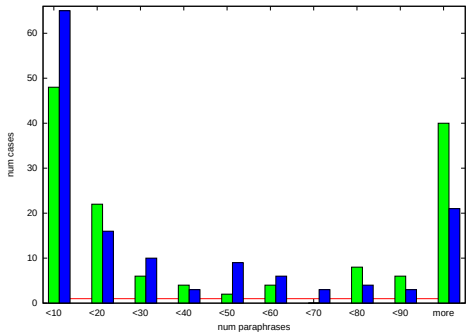

Figure 3: Distribution of generation outputs before and after debugging

It is very well to be cutting out overgeneration, but only so long as we are not cutting out linguistically valid sentences along the way. The test suite had been built semi-automatically, by parsing some sentences and hand-picking the valid semantic representations among the proposed outputs. As a basic sanity check, we reparsed the original sentences with the new grammar and found that 136 out of 140 sentences were parsed successfully, 4 less than with the original grammar. The difference was due to an over-restrictive constraint and was easily corrected.

4.2 Typing the suspects

As mentioned above, the overall 70% overgeneration reduction was achieved by a total of 13 modifications to the (meta)-grammar. Two points are worth stressing here.

First, the small number of modification is due to the fact that the metagrammar is a very compact description of the grammar where in particular, shared tree fragments are factored out and used in the production of several trees. As a result, one change to the metagrammar usually induces a change in not one but several (sometime hundred of) TAG trees. For instance, a modification stated in the fragment describing the verb spine of the active verb form will affect all trees in the grammar that realise an active verb form i.e., several hundreds of trees.

Second, the drastic reduction in overgeneration is made possible by a combination of 3 factors. First, the suspects report allows for a quick identification

of the overgeneration sources. Second, the metagrammar architecture makes it possible to generalise. Suppose for instance, that a given SR-ITEM indicates that the grammar incorrectly allows the adjunction of a given type of auxiliary tree β to a subject cleft tree. It might be the case that in fact, the grammar should be modified to block the combination of β with *all* cleft trees (not just the subject ones). Then the metagrammar architecture makes it possible to state the required modification at the level of the cleft description so that in effect, all cleft trees will be modified. In this way, the identification of an overgeneration cause linked to a specific example can be generalised to a larger class of examples. Third, the input data was organised in a graduated testsuite where first simple (basic) sentences where considered then sentences of complexity 2 (cases whose canonical verbalisation involve two finite verbs), then sentences of complexity 3 (three finite verbs). By proceeding incrementally through the testsuite, we ensured that early modifications propagate to more complex cases.

Let us now look at the types of errors which, we found, induce overgeneration.

Missing constraints Unsurprisingly, the main source of overgeneration was the lack of sufficient constraints to block illicit tree combinations. For instance, the grammar overgenerated the string *devoir c'est Jean qui part* (instead of *c'est Jean qui doit partir*) because the tree for *devoir* was not sufficiently constrained to block adjunction on the VP node of cleft trees. In such cases, adding the relevant constraints (e.g., CEST = - on the foot node of the *devoir*-tree and CEST = + on the VP node of the cleft-tree for *partir*) eliminates the overgeneration.

Incomplete constraints and incorrect feature percolation In some cases, we found that the constraint was only partially encoded by the grammar in that it was correctly stated in one of the combining trees but incorrectly or not at all in the other. Thus for instance, the adjective tree was correctly constrained to adjoin to DET = - N-trees but the corresponding DET = + constraint on the root node of determiner trees was missing. In other cases, the feature was present but incorrectly percolated. In both cases, the partial implementation of the constraint lead to a lack of unification clash and thereby to an overgenerating combination of trees.

Illicit elementary trees A third type of errors was linked to the fact that the grammar was produced semi-automatically from an abstract grammar description. In some cases, the linguist had failed to correctly foresee the implications of her description so that an elementary tree was produced by the compiler that was in fact incorrect. For instance, we had to introduce an additional constraint in the metagrammar to rule out the formation of trees describing a transitive verb with impersonal subject (in French, transitive verb cannot take an impersonal subject).

Incorrect semantics A more complex type of error to deal with concern cases where the semantics is insufficiently constrained thereby allowing for illicit combinations. For instance, in the imperative form, the grammar failed to constrain the first semantic argument to be YOU i.e., the hearer denotation. As a result, the input for sentences such as *Jean demande si Paul part* incorrectly generated strings such as *demande à Jean si Paul part*. In such cases thus, it is the semantics associated by the metagrammar with the elementary tree that needs to be modified.

Lexical exceptions As is well known, grammatical generalisation often are subject to lexical exceptions. For instance, transitive verbs are generally assumed to passivise but verbs of measure such as *to weigh* are transitive and do not. As is usual in TAG, in GENI, such exceptions are stated in the lexicon thereby blocking the selection of certain trees (in this case, all the passive trees) for the lexical items creating the exception (here the measure type verbs). Relatedly, some of the overgeneration cases stem from insufficient lexical information.

5 Conclusion

Debugging grammars for overgeneration need not be slow and tedious. We have found that with a certain dose of automation – a test harness to mechanise the regression-testing parts of the process, and computer-generated summaries to identify trouble spots – we can obtain major reductions in overgeneration with little effort.

Whilst these initial results are encouraging, a more sophisticated approach should help to detect more errors more efficiently. One shortcoming of our current approach is that we focus mostly on unitary sources of overgeneration: a single lexical item, tree property or derivation operation that consistently occurs in overgenerated strings. However, grammar flaws essentially consist of unexpected interactions between (at least) two items, so it would

seem that the most sensible place to look for mistakes would be where they interact. For example, instead of identifying single items that fail, we could look for *pairs* of items that consistently overgenerate when they co-occur. Note that this is not necessarily a subset of single-source failures. A given item X may consistently overgenerate in the presence of another item Y, but not with Z. If we were only looking for consistent single-source failures, we would ignore X altogether, whereas if we were looking for pairs, we would indeed detect (X,Y).

Another shortcoming of our approach is that it requires us to be disciplined in our pass/overgeneration annotations. If we mismark a sentence as pass, the derivation summariser will neglect every tree property or derivation item that occurs in that sentence, as it is only looking for items that consistently overgenerate. Perhaps a more robust approach would be to instead return items that *tend* to occur with overgeneration. This would make it more tolerant to imperfect annotations.

Producing these annotations is time-consuming. It would be worthwhile to explore some automatic means of making pass/overgeneration judgements on a large number of sentences, for example, using an n-gram based language model, like one that would be employed by a speech recogniser. We could then take the best N% of the sentences as passes or establish a threshold of improbability, below which sentences will be considered as overgeneration. We could also use more sophisticated tools, a statistical parser or a symbolic one with a wide coverage grammar in an alternate formalism. Even a relatively liberal parser which itself overgenerates might be useful in that (i) it may overgenerate in different areas than our grammar (ii) anything that *it* marks as a failure would be highly suspicious indeed.

The annotations do not need to be produced by a full-fledged parser either. Indeed, for each sentence that it produces, the surface realiser outputs its parse tree. So another way to classify the generated strings might be through assessing not the quality of the strings themselves but of their parses. For example, we could determine if the elementary trees that were used to build a sentence are likely to occur together in the same sentence. This kind of information can be extracted from a systemic functional grammar for instance. SFGs are largely generation-oriented grammars which encode as a network, the motivations behind each linguistic choice and the linguistic choices they allow for. If we associated each choice in the SFG network with a set of tree properties from our TAG grammar, we would essentially have an encoding of what tree properties go together. If the sentence contains a set of tree properties for which there is no equivalent system network traversal, it should be flagged as suspicious.

Our use of this test harness has so far been limited to the syntactic aspects of surface realisation. It could also be applied to other realiser tasks such as, for instance, morphological generation. It would also be interesting to see to what extent the method used here to spot overgeneration suspects could be adapted to other linguistic formalisms such as HPSG, LFG or CCG.

Finally, it would be interesting to investigate in how much overgeneration reduction helps reduce parsing ambiguity. Given a large scale symbolic grammar, parsing will often yield several hundreds of parses many of them are probably incorrect. We believe that reducing overgeneration should help reduce the number of output parses and thereby improve both parsing efficiency and the quality of the output parses.

GENI is free (GPL) software and can be downloaded at `http://trac.loria.fr/~geni`.

References

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

C. Gardent. Integration d'une dimension semantique dans les grammaires d'arbres adjoints. *TALN*, 2006.

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

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.

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