A Symbolic Approach to Near-Deterministic Surface Realisation using Tree Adjoining Grammar

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

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

Abstract

Surface realisers divide into those used in generation (NLG geared realisers) and those mirroring the parsing process (Reversible realisers). While the first rely on grammars not easily usable for parsing, it is unclear how the second type of realisers could be parameterised to yield from among the set of possible paraphrases, the paraphrase appropriate to a given generation context. In this paper, we present a surface realiser which combines a reversible grammar (used for parsing and doing semantic construction) with a symbolic means of selecting paraphrases.

1 Introduction

In generation, the *surface realisation* task consists in mapping a semantic representation into a grammatical sentence.

Depending on their use, on their degree of non-determinism and on the type of grammar they assume, existing surface realisers can be divided into two main categories namely, *NLG (Natural Language Generation) geared realisers* and *reversible realisers*.

NLG geared realisers are meant as modules in a full-blown generation system and as such, they are constrained to be deterministic: a generation system must output exactly one text, no less, no more. In order to ensure this determinism, NLG geared realisers generally rely on theories of grammar which systematically link form to function such as systemic functional grammar (SFG, (Matthiessen and Bateman, 1991)) and, to a lesser extent, Meaning Text Theory (MTT, (Mel'cuk, 1988)). In these theories, a sentence is associated not just with a semantic representation but with a semantic representation enriched with additional syntactic, pragmatic and/or discourse information. This additional information is then used to constrain the realiser output.[1] One drawback of these NLG geared realisers however, is that the grammar used is not usually reversible i.e., cannot be used both for parsing and for generation. Given the time and expertise involved in developing a grammar, this is a non-trivial drawback.

Reversible realisers on the other hand, are meant to mirror the parsing process. They are used on a grammar developed for parsing and equipped with a compositional semantics. Given a string and such a grammar, a parser will assign the input string all the semantic representations associated with that string by the grammar. Conversely, given a semantic representation and the same grammar, a realiser will assign the input semantics all the strings associated with that semantics by the grammar. In such approaches, non-determinism is usually handled by statistical filtering: treebank induced probabilities are used to select from among the possible paraphrases, the most probable one. Since the most probable paraphrase is not necessarily the most appropriate one in a given context, it is unclear however, how such realisers could be integrated into a generation system.

In this paper, we present a surface realiser which

[1] On the other hand, one of our reviewers noted that "determinism" often comes more from defaults when input constraints are not supplied. One might see these realisers as being less deterministic than advertised; however, the point is that it is possible to supply the constraints that ensure determinism.

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 328–335,
Prague, Czech Republic, June 2007.

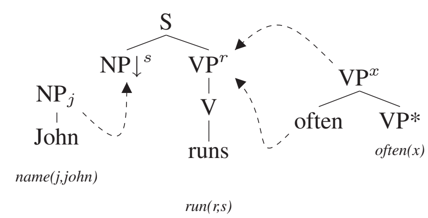
combines reversibility with a symbolic approach to determinism. The grammar used is fully reversible (it is used for parsing) and the realisation algorithm can be constrained by the input so as to ensure a unique output conforming to the requirement of a given (generation) context. We show both that the grammar used has a good paraphrastic power (it is designed in such a way that grammatical paraphrases are assigned the same semantic representations) and that the realisation algorithm can be used either to generate all the grammatical paraphrases of a given input or just one provided the input is adequately constrained.

The paper is structured as follows. Section 2 introduces the grammar used namely, a Feature Based Lexicalised Tree Adjoining Grammar enriched with a compositional semantics. Importantly, this grammar is compiled from a more abstract specification (a so-called "meta-grammar") and as we shall see, it is this feature which permits a natural and systematic coupling of semantic literals with syntactic annotations. Section 3 defines the surface realisation algorithm used to generate sentences from semantic formulae. This algorithm is non-deterministic and produces all paraphrases associated by the grammar with the input semantics. We then go on to show (section 4) how this algorithm can be used on a semantic input enriched with syntactic or more abstract control annotations and further, how these annotations can be used to select from among the set of admissible paraphrases precisely these which obey the constraints expressed in the added annotations. Section 5 reports on a quantitative evaluation based on the use of a core tree adjoining grammar for French. The evaluation gives an indication of the paraphrasing power of the grammar used as well as some evidence of the deterministic nature of the realiser. Section 6 relates the proposed approach to existing work and section 7 concludes with pointers for further research.

2 The grammar

We use a unification based version of LTAG namely, Feature-based TAG. A Feature-based TAG (FTAG, (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 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 (Gardent and Kallmeyer, 2003).



⇒ *name(j,john), run(r,j), often(r)*

Figure 1: Flat Semantics for "John often runs"

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

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

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 of 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).

3 The surface realiser, GenI

The basic surface realisation algorithm used is a bottom up, tabular realisation algorithm (Kay, 1996) 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*.

4 Paraphrase selection

The surface realisation algorithm just sketched is non-deterministic. Given a semantic formula, it might produce several outputs. For instance, given the appropriate grammar for French, the input in (2a) will generate the set of paraphrases partly given in (2b-2k).

(2) a. l_j*:jean(j)* l_a*:aime(e,j,m)* l_m*:marie(m)*
b. Jean aime Marie
c. Marie est aimée par Jean
d. C'est Jean qui aime Marie
e. C'est Jean par qui Marie est aimée
f. C'est par Jean qu'est aimée Marie
g. C'est Jean dont est aimée Marie
h. C'est Jean dont Marie est aimée
i. C'est Marie qui est aimée par Jean

j. C'est Marie qu'aime Jean
k. C'est Marie que Jean aime

To select from among all possible paraphrases of a given input, exactly one paraphrase, NLG geared realisers use symbolic information to encode syntactic, stylistic or pragmatic constraints on the output. Thus for instance, both REALPRO (Lavoie and Rambow, 1997) and SURGE (Elhadad and Robin, 1999) assume that the input associates semantic literals with low level syntactic and lexical information mostly leaving the realiser to just handle inflection, word order, insertion of grammatical words and agreement. Similarly, KPML (Matthiessen and Bateman, 1991) assumes access to ideational, interpersonal and textual information which roughly corresponds to semantic, mood/voice, theme/rheme and focus/ground information.

In what follows, we first show that the semantic input assumed by the realiser sketched in the previous section can be systematically enriched with syntactic information so as to ensure determinism. We then indicate how the satisfiability of this enriched input could be controlled.

4.1 At most one realisation

In the realisation algorithm sketched in Section 3, non-determinism stems from lexical ambiguity:[3] for each (combination of) literal(s) l in the input there usually is more than one TAG elementary tree whose semantics subsumes l. Thus each (combination of) literal(s) in the input selects a set of elementary trees and the realiser output is the set of combinations of selected lexical trees which are licensed by the grammar operations (substitution and adjunction) and whose semantics is the input.

One way to enforce determinism consists in ensuring that each literal in the input selects exactly one elementary tree. For instance, suppose we want to generate (2b), repeated here as (3a), rather than any of the paraphrases listed in (2c-2k). Intuitively, the syntactic constraints to be expressed are those given in (3b).

(3) a. Jean aime Marie
b. Canonical Nominal Subject, Active verb form, Canonical Nominal Object
c. l_j:*jean(j)* l_a:*aime(e,j,m)* l_m:*marie(m)*

The question is how precisely to formulate these constraints, how to associate them with the semantic input assumed in Section 3 and how to ensure that the constraints used do enforce uniqueness of selection (i.e., that for each input literal, exactly one elementary tree is selected)? To answer this, we rely on a feature of the grammar used, namely that *each elementary tree is associated with a linguistically meaningful unique identifier.*

The reason for this is that the grammar is compiled from a higher level description where tree fragments are first encapsulated into so-called classes and then explicitly combined (by inheritance, conjunction and disjunction) to produce the grammar elementary trees (cf. (Crabbé and Duchier, 2004)).

More generally, each elementary tree in the grammar is associated with the set of classes used to produce that tree and importantly, this set of classes (we will call this the *tree identifier*) provides a distinguishing description (a unique identifier) for that tree: a tree is defined by a specific combination of classes and conversely, a specific combination of classes yields a unique tree.[4] Thus the set of classes associated by the compilation process with a given elementary tree can be used to uniquely identify that tree.

Given this, surface realisation is constrained as follows.

1. Each tree identifier *Id(tree)* is mapped into a simplified set of tree properties TP_t. There are two reasons for this simplification. First, some classes are irrelevant. For instance, the class used to enforce subject-verb agreement is needed to ensure this agreement but does not help in selecting among competing trees. Second, a given class C can be defined to be

[3]Given two TAG trees, there might also be several ways of combining them thereby inducing more non-determinism. However in practice we found that most of this non-determinism is due either to over-generation (cases where the grammar is not sufficiently constrained and allows for one tree to adjoin to another tree in several places) or to spurious derivation (distinct derivations with identical semantics). The few remaining cases that are linguistically correct are due to varying modifier positions and could be constrained by a sophisticated feature decorations in the elementary tree.

[4]This is not absolutely true as a tree identifier only reflects part of the compilation process. In practice, they are few exceptions though so that distinct trees whose tree identifiers are identical can be manually distinguished.

equivalent to the combination of other classes $C_1 \ldots C_n$ and consequently a tree identifier containing $C, C_1 \ldots C_n$ can be reduced to include either C or $C_1 \ldots C_n$.

2. Each literal l_i in the input is associated with a tree property set TP_i (i.e., the input we generate from is enriched with syntactic information)

3. During realisation, for each literal/tree property pair $\langle l_i : TP_i \rangle$ in the enriched input semantics, lexical selection is constrained to retrieve only those trees (i) whose semantics subsumes l_i and (ii) whose tree properties are TP_i

Since each literal is associated with a (simplified) tree identifier and each tree identifier uniquely identifies an elementary tree, realisation produces at most one realisation.

Examples 4a-4c illustrates the kind of constraints used by the realiser.

(4) a. l_j*:jean(j)/ProperName*
l_a*:aime(e,j,m)/[CanonicalNominalSubject, ActiveVerbForm, CanonicalNominalObject]*
l_m*:marie(m)/ProperName*
Jean aime Marie
* Jean est aimé de Marie

b. l_c*:le(c)/Det*
l_c*:chien(c)/Noun*
l_d*:dort(e1,c)/RelativeSubject*
l_r*:ronfle(e2,c)/CanonicalSubject*
Le chien qui dort ronfle
* Le chien qui ronfle dort

c. l_j*:jean(j)/ProperName*
l_p*:promise(e1,j,m,e2)/[CanonicalNominalSubject, ActiveVerbForm, CompletiveObject]*
l_m*:marie(m)/ProperName*
l_{e2}*:partir(e2,j)/InfinitivalVerb*
Jean promet à marie de partir
* Jean promet à marie qu'il partira

4.2 At least one realisation

For a realiser to be usable by a generation system, there must be some means to ensure that its input is satisfiable i.e., that it can be realised. How can this be done without actually carrying out realisation i.e., without checking that the input is satisfiable? Existing realisers indicate two types of answers to that dilemma.

A first possibility would be to draw on (Yang et al., 1991)'s proposal and compute the enriched input based on the traversal of a systemic network. More specifically, one possibility would be to consider a systemic network such as NIGEL, precompile all the functional features associated with each possible traversal of the network, map them onto the corresponding tree properties and use the resulting set of tree properties to ensure the satisfiability of the enriched input.

Another option would be to check the well formedness of the input at some level of the linguistic theory on which the realiser is based. Thus for instance, REALPRO assumes as input a well formed deep syntactic structure (DSyntS) as defined by Meaning Text Theory (MTT) and similarly, SURGE takes as input a functional description (FD) which in essence is an underspecified grammatical structure within the SURGE grammar. In both cases, there is no guarantee that the input be satisfiable since all the other levels of the linguistic theory must be verified for this to be true. In MTT, the DSyntS must first be mapped onto a surface syntactic structure and then successively onto the other levels of the theory while in SURGE, the input FD can be realised only if it provides consistent information for a complete top-down traversal of the grammar right down to the lexical level. In short, in both cases, the well formedness of the input can be checked with respect to some criteria (e.g., well formedness of a deep syntactic structure in MTT, well formedness of a FD in SURGE) but this well formedness does not guarantee satisfiability. Nonetheless this basic well formedness check is important as it provides some guidance as to what an acceptable input to the realiser should look like.

We adopt a similar strategy and resort to the notion of *polarity neutral input* to control the well formedness of the enriched input. The proposal draws on ideas from (Koller and Striegnitz, 2002; Gardent and Kow, 2005) and aims to determine whether for a given input (a set of TAG elementary trees whose semantics equate the input semantics), syntactic requirements and resources cancel out. More specifically, the aim is to determine whether given the input set of elementary trees, each substitution and each adjunction requirement is satisfied by exactly one elementary tree of the appropriate syntactic category and semantic index.

Roughly,[5] the technique consists in (automatically) associating with each elementary tree a polarity signature reflecting its substitution/adjunction requirements and resources and in computing the grand polarity of each possible combination of trees covering the input semantics. Each such combination whose total polarity is non-null is then filtered out (not considered for realisation) as it cannot possibly lead to a valid derivation (either a requirement cannot be satisfied or a resource cannot be used).

In the context of a generation system, polarity checking can be used to check the satisfiability of the input or more interestingly, to correct an ill formed input i.e., an input which can be detected as being unsatisfiable.

To check a given input, it suffices to compute its polarity count. If it is non-null, the input is unsatisfiable and should be revised. This is not very useful however, as the enriched input ensures determinism and thereby make realisation very easy, indeed almost as easy as polarity checking.

More interestingly, polarity checking can be used to suggest ways of fixing an ill formed input. In such a case, the enriched input is stripped of its control annotations, realisation proceeds on the basis of this simplified input and polarity checking is used to preselect all polarity neutral combinations of elementary trees. A closest match (i.e. the polarity neutral combination with the greatest number of control annotations in common with the ill formed input) to the ill formed input is then proposed as a probably satisfiable alternative.

5 Evaluation

To evaluate both the paraphrastic power of the realiser and the impact of the control annotations on non-determinism, we used a graduated test-suite which was built by (i) parsing a set of sentences, (ii) selecting the correct meaning representations from the parser output and (iii) generating from these meaning representations. The gradation in the test suite complexity was obtained by partitioning the input into sentences containing one, two or three finite verbs and by choosing cases allowing for different paraphrasing patterns. More specifically, the test suite includes cases involving the following types of paraphrases:

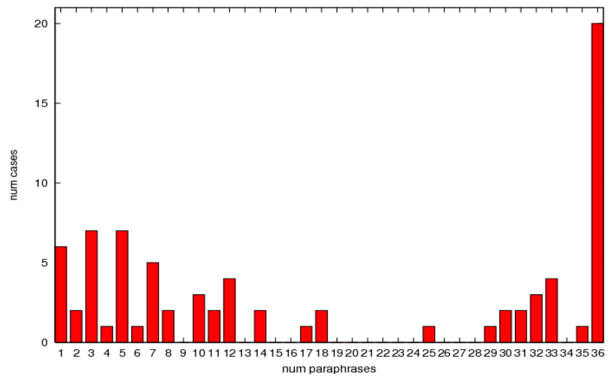
- Grammatical variations in the realisations of the arguments (cleft, cliticisation, question, relativisation, subject-inversion, etc.) or of the verb (active/passive, impersonal)
- Variations in the realisation of modifiers (e.g., relative clause vs adjective, predicative vs non-predicative adjective)
- Variations in the position of modifiers (e.g., pre- vs post-nominal adjective)
- Variations licensed by a morpho-derivational link (e.g., to arrive/arrival)

On a test set of 80 cases, the paraphrastic level varies between 1 and over 50 with an average of 18 paraphrases per input (taking 36 as upper cut off point in the paraphrases count). Figure 5 gives a more detailed description of the distribution of the paraphrastic variation. In essence, 42% of the sentences with one finite verb accept 1 to 3 paraphrases (cases of intransitive verbs), 44% accept 4 to 28 paraphrases (verbs of arity 2) and 13% yield more than 29 paraphrases (ditransitives). For sentences containing two finite verbs, the ratio is 5% for 1 to 3 paraphrases, 36% for 4 to 14 paraphrases and 59% for more than 14 paraphrases. Finally, sentences containing 3 finite verbs all accept more than 29 paraphrases.

Two things are worth noting here. First, the paraphrase figures might seem low wrt to e.g., work by (Velldal and Oepen, 2006) which mentions several thousand outputs for one given input and an average number of realisations per input varying between 85.7 and 102.2. Admittedly, the French grammar we are using has a much more limited coverage than the ERG (the grammar used by (Velldal and Oepen, 2006)) and it is possible that its paraphrastic power is lower. However, the counts we give only take into account *valid paraphrases of the input*. In other words, overgeneration and spurious derivations are excluded from the toll. This does not seem to be the case in (Velldal and Oepen, 2006)'s approach where the count seems to include *all* sentences associated by the grammar with the input semantics.

Second, although the test set may seem small it is important to keep in mind that it represents 80 inputs

[5] Lack of space prevents us from giving much details here. We refer the reader to (Koller and Striegnitz, 2002; Gardent and Kow, 2005) for more details.



with distinct grammatical and paraphrastic properties. In effect, these 80 test cases yields 1 528 distinct well-formed sentences. This figure compares favourably with the size of the largest regression test suite used by a symbolic NLG realiser namely, the SURGE test suite which contains 500 input each corresponding to a single sentence. It also compares reasonably with other more recent evaluations (Callaway, 2003; Langkilde-Geary, 2002) which derive their input data from the Penn Treebank by transforming each sentence tree into a format suitable for the realiser (Callaway, 2003). For these approaches, the test set size varies between roughly 1 000 and almost 3 000 sentences. But again, it is worth stressing that these evaluations aim at assessing coverage and correctness (does the realiser find the sentence used to derive the input by parsing it?) rather than the paraphrastic power of the grammar. They fail to provide a systematic assessment of how many distinct grammatical paraphrases are associated with each given input.

To verify the claim that tree properties can be used to ensure determinism (cf. footnote 4), we started by eliminating from the output all ill-formed sentences. We then automatically associated each well-formed output with its set of tree properties. Finally, for each input semantics, we did a systematic pairwise comparison of the tree property sets associated with the input realisations and we checked whether for any given input, there were two (or more) distinct paraphrases whose tree properties were the same. We found that such cases represented slightly over 2% of the total number of (input,realisations) pairs. Closer investigation of the faulty data indicates two main reasons for non-determinism namely, trees with alternating order of arguments and derivations with distinct modifier adjunctions. Both cases can be handled by modifying the grammar in such a way that those differences are reflected in the tree properties.

6 Related work

The approach presented here combines a reversible grammar realiser with a symbolic approach to paraphrase selection. We now compare it to existing surfaces realisers.

NLG geared realisers. Prominent general purpose NLG geared realisers include REALPRO, SURGE, KPML, NITROGEN and HALOGEN. Furthermore, HALOGEN has been shown to achieve broad coverage and high quality output on a set of 2 400 input automatically derived from the Penn treebank.

The main difference between these and the present approach is that our approach is based on a reversible grammar whilst NLG geared realisers are not. This has several important consequences.

First, it means that one and *the same grammar and lexicon can be used both for parsing and for generation*. Given the complexity involved in developing such resources, this is an important feature.

Second, as demonstrated in the Redwood Lingo Treebank, reversibility makes it easy to *rapidly create very large evaluation suites*: it suffices to parse a set of sentences and select from the parser output the correct semantics. In contrast, NLG geared realisers either work on evaluation sets of restricted size (500 input for SURGE, 210 for KPML) or require the time expensive implementation of a preprocessor transforming e.g., Penn Treebank trees into a format suitable for the realisers. For instance, (Callaway, 2003) reports that the implementation of such a processor for SURGE was the most time consuming part of the evaluation with the resulting component containing 4000 lines of code and 900 rules.

Third, a reversible grammar can be exploited to *support not only realisation but also its reverse, namely semantic construction*. Indeed, reversibility is ensured through a compositional semantics that is, through a tight coupling between syntax and semantics. In contrast, NLG geared realisers often have to reconstruct this association in rather ad hoc ways. Thus for instance, (Yang et al., 1991) resorts to ad hoc “mapping tables” to associate substitution nodes with semantic indices and “fr-nodes” to constrain adjunction to the correct nodes. More generally, the lack of a clearly defined compositional semantics in NLG geared realisers makes it difficult to see how the grammar they use could be exploited to also support semantic construction.

Fourth, the *grammar can be used both to generate and to detect paraphrases*. It could be used for instance, in combination with the parser and the semantic construction module described in (Gardent and Parmentier, 2005), to support textual entailment recognition or answer detection in question answering.

Reversible realisers. The realiser presented here differs in mainly two ways from existing reversible realisers such as (White, 2004)'s CCG system or the HPSG ERG based realiser (Carroll and Oepen, 2005).

First, it permits a symbolic selection of the output paraphrase. In contrast, existing reversible realisers use statistical information to select from the produced output the most plausible paraphrase.

Second, particular attention has been paid to the treatment of paraphrases in the grammar. Recall that TAG elementary trees are grouped into families and further, that the specific TAG we use is compiled from a highly factorised description. We rely on these features to associate one and the same semantic to large sets of trees denoting semantically equivalent but syntactically distinct configurations (cf. (Gardent, 2006)).

7 Conclusion

The realiser presented here, GENI, exploits a grammar which is produced semi-automatically by compiling a high level grammar description into a Tree Adjoining Grammar. We have argued that a side-effect of this compilation process – namely, the association with each elementary tree of a set of tree properties – can be used to constrain the realiser output. The resulting system combines the advantages of two orthogonal approaches. From the reversible approach, it takes the reusability, the ability to rapidly create very large test suites and the capacity to both generate and detect paraphrases. From the NLG geared paradigm, it takes the ability to symbolically constrain the realiser output to a given generation context.

GENI is free (GPL) software and is available at `http://trac.loria.fr/˜geni`.

References

Charles B. Callaway. 2003. Evaluating coverage for large symbolic NLG grammars. In *18th IJCAI*, pages 811–817, Aug.

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

B. Crabbé and D. Duchier. 2004. Metagrammar redux. In *CSLP*, Copenhagen.

M. Elhadad and J. Robin. 1999. SURGE: a comprehensive plug-in syntactic realization component for text generation. *Computational Linguistics*.

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

C. Gardent and E. Kow. 2005. Generating and selecting grammatical paraphrases. *ENLG*, Aug.

C. Gardent and Y. Parmentier. 2005. Large scale semantic construction for Tree Adjoining Grammars. *LACL05*.

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

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

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

I. Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the INLG*.

B. Lavoie and O. Rambow. 1997. RealPro–a fast, portable sentence realizer. *ANLP'97*.

C. Matthiessen and J.A. Bateman. 1991. *Text generation and systemic-functional linguistics: experiences from English and Japanese*. Frances Pinter Publishers and St. Martin's Press, London and New York.

I.A. Mel'cuk. 1988. *Dependency Syntax: Theorie and Practice*. State University Press of New York.

Erik Velldal and Stephan Oepen. 2006. Statistical ranking in tactical generation. In *EMNLP*, Sydney, Australia.

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

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

G. Yang, K. McKoy, and K. Vijay-Shanker. 1991. From functional specification to syntactic structure. *Computational Intelligence*, 7:207–219.