

Generation for Grammar Engineering

Claire Gardent, German Kruszewski

▶ **To cite this version:**

Claire Gardent, German Kruszewski. Generation for Grammar Engineering. INLG 2012, The seventh International Natural Language Generation Conference., May 2012, Starved Rock, Illinois, United States. pp.31-40. hal-00768612

HAL Id: hal-00768612
https://hal.science/hal-00768612

Submitted on 22 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Generation for Grammar Engineering

Claire Gardent
CNRS, LORIA, UMR 7503
Vandoeuvre-lès-Nancy, F-54000, France
claire.gardent@loria.fr

German Kruszewski
Inria, LORIA, UMR 7503
Villers-lès-Nancy, F-54600, France
german.kruszewski@inria.fr

Abstract

While in Computer Science, grammar engineering has led to the development of various tools for checking grammar coherence, completion, under- and over-generation, in Natural Langage Processing, most approaches developed to improve a grammar have focused on detecting under-generation and to a much lesser extent, over-generation. We argue that generation can be exploited to address other issues that are relevant to grammar engineering such as in particular, detecting grammar incompleteness, identifying sources of over-generation and analysing the linguistic coverage of the grammar. We present an algorithm that implements these functionalities and we report on experiments using this algorithm to analyse a Feature-Based Lexicalised Tree Adjoining Grammar consisting of roughly 1500 elementary trees.

1 Introduction

Grammar engineering, the task of developing large scale computational grammars, is known to be error prone. As the grammar grows, the interactions between the rules and the lexicon become increasingly complex and the generative power of the grammar becomes increasingly difficult for the grammar writer to predict.

While in Computer Science, grammar engineering has led to the development of various tools for checking grammar coherence, completion, under- and over-generation (Klint et al., 2005), in Natural Langage Processing, most approaches developed to improve a grammar have focused on detecting under-generation (that is cases where the grammar and/or the lexicon fails to provide an analysis for a given, grammatical, input) and to a lesser degree over-generation.

In this paper, we argue that generation can be exploited to address other issues that are relevant to grammar engineering. In particular, we claim that it can be used to:

- Check grammar completeness: for each grammar rule, is it possible to derive a syntactically complete tree ? That is, can each grammar rule be used to derive a constituent.

- Analyse generation and over-generation: given some time/recursion upper bounds, what does the grammar generate? How much of the output is over-generation? Which linguistic constructions present in a language are covered by the grammar?

We present a generation algorithm called GRADE (GRAmmar DEbugger) that permits addressing these issues. In essence, this algorithm implements a top-down grammar traversal guided with semantic constraints and controlled by various parameterisable constraints designed to ensure termination and linguistic control.

The GRADE algorithm can be applied to any generative grammar i.e., any grammar which uses a start symbol and a set of production rules to generate the sentences of the language described by that grammar. We present both an abstract description of this algorithm and a concrete implementation which takes advantage of Definite Clause Grammars

INLG 2012 Proceedings of the 7th International Natural Language Generation Conference, pages 31–39,
Utica, May 2012. ©2012 Association for Computational Linguistics

to implement grammar traversal. We then present the results of several experiments where we use the GRADE algorithm to examine the output of SEMTAG, a Feature-Based Lexicalised Tree Adjoining Grammar (FB-LTAG) for French.

The paper is structured as follows. Section 2 summarises related work. Section 3 presents the GRADE algorithm. Section 4 introduces the grammar used for testing and describes an implementation of GRADE for FB-LTAG. Section 5 presents the results obtained by applying the GRADE algorithm to SEMTAG. We show that it helps (i) to detect sources of grammar incompleteness (i.e., rules that do not lead to a complete derivation) and (ii) to identify overgeneration and analyse linguistic coverage. Section 6 concludes.

2 Related Work

Two main approaches have so far been used to improve grammars: treebank-based evaluation and error mining techniques. We briefly review this work focusing first, on approaches that are based on parsing and second, on those that exploit generation.

Debugging Grammars using Parsing Over the last two decades, *Treebank-Based evaluation* has become the standard way of evaluating parsers and grammars. In this framework (Black et al., 1991), the output of a parser is evaluated on a set of sentences that have been manually annotated with their syntactic parses. Whenever the parse tree produced by the parser differs from the manual annotation, the difference can be traced back to the parser (timeout, disambiguation component), the grammar and/or to the lexicon. Conversely, if the parser fails to return an output, undergeneration can be traced back to missing or erroneous information in the grammar or/and in the lexicon.

While it has supported the development of robust, large coverage parsers, treebank based evaluation is limited to the set of syntactic constructions and lexical items present in the treebank. It also fails to directly identify the most likely source of parsing failures. To bypass these limitations, *error mining techniques* have been proposed which permit detecting grammar and lexicon errors by parsing large quantities of data (van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok et al., 2009). The output of this parsing process is then divided into two sets of parsed and unparsed sentences which are used to compute the "suspicion rate" of n-grams of word forms, lemmas or part of speech tags whereby the suspicion rate of an item indicates how likely a given item is to cause parsing to fail. Error mining was shown to successfully help detect errors in the lexicon and to a lesser degree in the grammar.

Debugging Grammars using Generation Most of the work on treebank-based evaluation and error mining target undergeneration using parsing. Recently however, some work has been done which exploits generation and more specifically, surface realisation to detect both under- and over-generation.

Both (Callaway, 2003) and the Surface Realisation (SR) task organised by the Generation Challenge (Belz et al., 2011) evaluate the output of surface realisers on a set of inputs derived from the Penn Treebank. As with parsing, these approaches permit detecting under-generation in that an input for which the surface realiser fails to produce a sentence points to shortcomings either in the surface realisation algorithm or in the grammar/lexicon. The approach also permits detecting overgeneration in that a low BLEU score points to these inputs for which the realiser produced a sentence that is markedly different from the expected answer.

Error mining approaches have also been developed using generation. (Gardent and Kow, 2007) is similar in spirit to the error mining approaches developed for parsing. Starting from a set of manually defined semantic representations, the approach consists in running a surface realiser on these representations; manually sorting the generated sentences as correct or incorrect; and using the resulting two datasets to detect grammatical structures that systematically occur in the incorrect dataset. The approach however is only partially automatised since both the input and the output need to be manually produced/annotated. More recently, (Gardent and Narayan, 2012) has shown how the fully automatic error mining techniques used for parsing could be adapted to mine for errors in the output of a surface realiser tested on the SR input data. In essence, they present an algorithm which enumerate the subtrees in the input data that frequently occur in surface realisation failure (the surface realiser fails to gener-

ate a sentence) and rarely occur in surface realisation success. In this way, they can identify subtrees in the input that are predominantly associated with generation failure.

In sum, tree-bank based evaluation permits detecting over- and under-generation while error mining techniques permits identifying sources of errors; Treebank-based evaluation requires a reference corpus while error mining techniques require a way to sort good from bad ouput; and in all cases, generation-based grammar debugging requires input to be provided (while for parsing, textual input is freely available).

Discussion The main difference between the GRADE approach and both error mining and treebank based evaluation is that GRADE is grammar based. No other input is required for the GRADE algorithm to work than the grammar[1]. Whereas existing approaches identify errors by processing large amounts of data, GRADE identifies errors by traversing the grammar. In other words, while other approaches assess the coverage of a parser or a generator on a given set of input data, GRADE permits systematically assessing the linguistic coverage and the precision of the constructs described by the grammar independently of any input data.

Currently, the output of GRADE needs to be manually examined and the sources of error manually identified. Providing an automatic means of sorting GRADE 's output into good and bad sentences is developed however, it could be combined with error mining techniques so as to facilitate interpretation.

3 The GraDE Algorithm

How can we explore the quirks and corners of a grammar to detect inconsistencies and incorrect output?

In essence, the GRADE algorithm performs a top-down grammar traversal and outputs the sentences generated by this traversal. It is grammar neutral in that it can be applied to any generative grammar i.e., any grammar which includes a start symbol and a set of production rules. Starting from the string consisting of the start symbol, the GRADE algorithm recursively applies grammar rules replacing one occurrence of its left-hand side in the string by its right-hand side until a string that contains neither the start symbol nor designated nonterminal symbols is produced.

Since NL grammars describe infinite sets of sentences however, some means must be provided to control the search and output sets of sentences that are linguistically interesting. Therefore, the GRADE algorithm is controlled by several user-defined parameters designed to address termination (Given that NL grammars usually describe an infinite set of sentences, how can we limit sentence generation to avoid non termination?), linguistic control (How can we control sentence generation so that the sentences produced cover linguistic variations that the linguist is interested in ?) and readibility (How can we constrain sentence generation in such a way that the output sentences are meaningful sentences rather than just grammatical ones?).

3.1 Ensuring termination

To ensure termination, GRADE supports three user-defined control parameters which can be used simultaneously or in isolation namely: a time out parameter; a restriction on the number and type of recursive rules allowed in any derivation; and a restriction on the depth of the derivation tree.

Each of these restrictions is implemented as a restriction on the grammar traversal process as follows.

Time out. The process halts when the time bound is reached.

Recursive Rules. For each type of recursive rule, a counter is created which is initialised to the values set by the user and decremented each time a recursive rule of the corresponding type is used. When all counters are null, recursive rules can no longer be used. The type of a recursive rule is simply the main category expanded by that rule namely, N, NP, V, VP and S. In addition, whenever a rule is applied, the GRADE algorithm arbitrarily divides up the recursion quotas of a symbol among the symbol's children. If it happens to divide them a way that cannot be fulfilled, then it fails, backtracks, and divides them some other way.

[1] Although some semantic input is possible.

Derivation Depth. A counter is used to keep track of the depth of the derivation tree and either halts (if no other rule applies) or backtracks whenever the set depth is reached.

3.2 Linguistic Coverage and Output Readibility

GRADE provides several ways of controlling the linguistic coverage and the readibility of the output sentences.

Modifiers. As we shall show in Section 5, the recursivity constraints mentioned in the previous section can be used to constrain the type and the number of modifiers present in the output.

Root Rule. Second, the "root rule" i.e., the rule that is used to expand the start symbol can be constrained in several ways. The user can specify which rule should be used; which features should label the lhs of that rule; which subcategorisation type it should model; and whether or not it is a recursive rule. For instance, given the FB-LTAG we are using, by specifying the root rule to be used, we can constrain the generated sentences to be sentences containing an intransitive verb in the active voice combining with a canonical nominal subject. If we only specify the subcategorisation type of the root rule e.g., transitive, we can ensure that the main verb of the generated sentences is a transitive verb; And if we only constrain the features of the root rule to indicative mode and active voice, then we allow for the generation of any sentence whose main verb is in the indicative mode and active voice.

Input Semantics. Third, in those cases where the grammar is a reversible grammar associating sentences with both a syntactic structure and a semantic representation, the content of the generated sentences can be controlled by providing GRADE with an input semantics. Whenever a core semantics is specified, only rules whose semantics includes one or more literal(s) in the core semantics can be used. Determiner rules however are selected independent of their semantics. In this way, it is possible to constrain the output sentences to verbalise a given meaning without having to specify their full semantics (the semantic representations used in reversible grammars are often intricate representations which are difficult to specify manually) and while allowing for morphological variations (tense, number, mode and aspect can be left unspecified and will be informed by the calls to the lexicon embedded in the DCG rules) as well as variations on determiners[2]. For instance, the core semantics {`run(E M)`, `man(M)`} is contained in, and therefore will generate, the flat semantics for the sentences *The man runs, The man ran, A man runs, A man ran, This man runs, My man runs, etc.*.

4 Implementation

In the previous section, we provided an abstract description of the GRADE algorithm. We now describe an implementation of that algorithm tailored for FB-LTAGs equipped with a unification-based compositional semantics. We start by describing the grammar used (SEMTAG), we then summarise the implementation of GRADE for FB-LTAG.

4.1 SemTAG

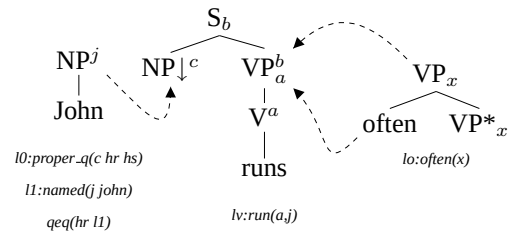
For our experiments, we use the FB-LTAG described in (Crabbé, 2005; Gardent, 2008). This grammar, called SEMTAG, integrates a unification-based semantics and can be used both for parsing and for generation. It covers the core constructs for non verbal constituents and most of the verbal constructions for French. The semantic representations built are MRSs (Minimal Recursion Semantic representations, (Copestake et al., 2001)).

More specifically, a tree adjoining grammar (TAG) is a tuple $\langle \Sigma, N, I, A, S \rangle$ with Σ a set of terminals, N a set of non-terminals, I a finite set of initial trees, A a finite set of auxiliary trees, and S a distinguished non-terminal ($S \in N$). Initial trees are trees whose leaves are labeled with substitution nodes (marked with a downarrow) or terminal categories[3]. Auxiliary trees are distinguished by a foot node (marked with a star) whose category must be the same as that of the root node.

[2]The rules whose semantics is not checked during derivation are specified as a parameter of the system and can be modified at will e.g., to include adverbs or auxiliaries. Here we choosed to restrict underspecification to determiners.

[3]Due to space limitation we here give a very sketchy definition of FB-LTAG. For a more detailed presentation, see (Vijay-Shanker and Joshi, 1988).

Two tree-composition operations are used to combine trees: substitution and adjunction. Substitution inserts a tree onto a substitution node of some other tree while adjunction inserts an auxiliary tree into a tree. In a Feature-Based Lexicalised TAG (FB-LTAG), tree nodes are furthermore decorated with two feature structures (called **top** and **bottom**) which are unified during derivation; and each tree is anchored with a lexical item. Figure 1 shows an example toy FB-LTAG with unification semantics.



$\Rightarrow$ *l0:proper_q(c hr hs) l1:named(j john), qeq(hr, l1), lv:run(a,j), lv:often(a)*

Figure 1: MRS for "John often runs"

4.2 GraDe for FB-LTAG

The basic FB-LTAG implementation of GRADE is described in detail in (Gardent et al., 2011; Gardent et al., 2010). In brief, this implementation takes advantage of the top-down, left-to-right, grammar traversal implemented in Definite Clause Grammars by translating the FB-LTAG to a DCG. In the DCG formalism, a grammar is represented as a set of Prolog clauses and Prolog's query mechanism provides a built-in top-down, depth-first, traversal of the grammar. In addition, the DCG formalism allows arbitrary Prolog goals to be inserted into a rule. To implement a controlled, top-down grammar traversal of SEMTAG, we simply convert SEMTAGto a Definite Clause Grammar (DCG) wherein arbitrary Prolog calls are used both to ground derivations with lexical items and to control Prolog's grammar traversal so as to respect the user defined constraints on recursion and on linguistic coverage. In addition, we extended the approach to handle semantic constraints (i.e., to allow for an input semantic to constrain the traversal) as discussed in Section 3. That is, for a subset of the grammar rules, a rule will only be applied if its semantics subsumes a literal in the input semantics.

For more details, on the FB-LTAG implementation of the GRADE algorithm and of the conversion from FB-LTAG to DCG, we refer the reader to (Gardent et al., 2011; Gardent et al., 2010).

5 Grammar Analysis

Depending on which control parameters are used, the GRADE algorithm can be used to explore the grammar from different viewpoints. In what follows, we show that it can be used to check grammar completeness (Can all rules in the grammar be used so as to derive a constituent?); to inspect the various possible realisations of syntactic functors and of their arguments (e.g., Are all possible syntactic realisations of the verb and of its arguments generated and correct?); to explore the interactions between basic clauses and modifiers; and to zoom in on the morphological and syntactic variants of a given core semantics (e.g., Does the grammar correctly account for all such variants ?).

5.1 Checking for Grammar Completeness

We first use GRADE to check, for each grammar rule, whether it can be used to derive a complete constituent i.e., whether a derivation can be found such that all leaves of the derivation tree are terminals (words). Can all trees anchored by a verb for instance, be completed to build a syntactically complete clause? Trees that cannot yield a complete constituent points to gaps or inconsistencies in the grammar.

To perform this check, we run the GRADE algorithm on verb rules, allowing for up to 1 adjunction on either a noun, a verb or a verb phrase and halting when either a derivation has been found or all possible rule combinations have been tried. Table 1 shows the results per verb family[4]. As can be seen, there are strong differences between the families with e.g., 80% of the trees failing to yield a derivation in the n0Vs1int (Verbs with interrogative sentential complement) family against 0% in the ilV

[4]The notational convention for verb types is from XTAG and reads as follows. Subscripts indicate the thematic role of the verb argument. n indicates a nominal, Pn a PP and s a sentential argument. pl is a verbal particle. Upper case letters describe the syntactic functor type: V is a verb, A an adjective and BE the copula. For instance, n0Vn1 indicates a verb taking two nominal arguments (e.g., *like*) .

Tree Family	Trees	Fails	Fails/Trees
CopulaBe	60	1	1%
ilV	2	0	0%
n0V	10	0	0%
n0ClV	9	0	0%
n0ClVn1	45	2	4%
n0ClVden1	36	3	8%
n0ClVpn1	29	3	10%
n0Vn1	84	3	3%
n0Vn1Adj2	24	6	25%
n0Van1	87	3	3%
n0Vden1	38	3	7%
n0Vpn1	30	3	10%
ilVcs1	2	0	0%
n0Vcs1	30	23	74%
n0Vas1	15	10	66%
n0Vn1Adj2	24	0	0%
s0Vn1	72	9	12%
n0Vs1int	15	12	80%
n0Vn1n2	24	0	0%
n0Vn1an2	681	54	7%

Table 1: Checking for Gaps in the Grammar

(impersonal with expletive subject, "it rains") and the n0V (intransitive, "Tammy sings"). In total, approximatively 10% (135/1317) of the grammar rules cannot yield a derivation.

5.2 Functor/Argument Dependencies

To check grammar completeness, we need only find one derivation for any given tree. To assess the degree to which the grammar correctly generates all possible realisations associated with a given syntactic functor however, all realisations generated by the grammar need to be produced. To restrict the output to sentences illustrating functor/argument dependencies (no modifiers), we constrain adjunction to the minimum required by each functor. In most cases, this boils down to setting the adjunction counters to null for all categories. One exception are verbs taking a sentential argument which require one S adjunction. We also allow for one N-adjunction and one V-adjunction to allow for determiners and the inverted subject clitic (t'il). In addition, the lexicon is restricted to avoid lexical or morphological variants.

We show below some of the well-formed sentences output by GRADE for the n0V (intransitive verbs) family.

> Elle chante (*She sings*), La tatou chante-t'elle? (*Does the armadillo sing?*), La tatou chante (*The armadillo sings*), La tatou qui chante (*The armadillo which sings*), Chacun chante -t'il (*Does everyone sing?*), Chacun chante (*Everyone sings*), Quand chante chacun? (*When does everyone sing?*), Quand chante la tatou? (*When does the armadillo sing?*) Quand chante quel tatou? (*When does which armadillo sing?*), Quand chante Tammy? (*When does Tammy sing?*), Chante-t'elle? (*Does she sing?*) Chante -t'il? (*Does he sing?*), Chante! (*Sing!*), Quel tatou chante ? (*Which armadillo sing?*), Quel tatou qui chante ..? (*Which armadillo who sings ..?*) Tammy chante-t'elle? (*Does Tammy sing?*), Tammy chante (*Tammy sings*), une tatou qui chante chante (*An armadillo which sings sings*), C'est une tatou qui chante (*It is an armadillo which sings*), ...

The call on this family returned 55 distinct MRSs and 65 distinct sentences of which only 28 were correct. Some of the incorrect cases are shown below. They illustrate the four main sources of overgeneration. The agreement between the inverted subject clitic and the subject fails to be enforced (a); the inverted nominal subject fails to require a verb in the indicative mode (b); the inverted subject clitic fails to be disallowed in embedded clauses (c); the interrogative determiner *quel* fails to constrain its nominal head to be a noun (d,e).

> (a) Chacun chante-t'elle? (*Everyone sings?*) (b) Chantée chacun? (*Sung everyone?*) (c) La tatou qui chante-t'elle? (*The armadillo which does she sing?*) (d) Quel chacun chante ? (*Which everyone sings?*) (e) quel tammy chante ? (*Which Tammy sings?*)

5.3 Interactions with Modifiers

Once basic functor/argument dependencies have been verified, adjunction constraints can be used to

explore the interactions between e.g., basic clauses and modification[5]. Allowing for N-adjunctions for instance, will produce sentences including determiners and adjectives. Similarly, allowing for V adjunction will permit for auxiliaries and adverbs to be used; and allowing for VP or S adjunctions will licence the use of raising verbs and verbs subcategorising for sentential argument.

We queried GRADE for derivations rooted in n0V (intransitive verbs) and with alternatively, 1N, 2N, 1V and 1VP adjunction. Again a restricted lexicon was used to avoid structurally equivalent but lexically distinct variants. The following table shows the number of sentences output for each query.

0	1S	1VP	1V	1N	2N
36	170	111	65	132	638

As the examples below show, the generated sentences unveil two further shortcomings in the grammar: the inverted subject clitic fails to be constrained to occur directly after the verb (1) and the order and compatibility of determiners are unrestricted (2).

(1) a. Semble-t'il chanter? / * Semble chanter t'il? (*Does he seems to sing?*)
   b. Chante-t'il dans Paris? / * Chante dans Paris-t'il? (*Does he sing in Paris?*)
   c. Chante-t'il beaucoup? / * Chante beaucoup-t'il? (*Does he sing a lot?*)
   d. Veut-t'il que Tammy chante? / * Veut que Tammy chante-t'il? (*Does he want that Tammy sings?*

(2) * Un quel tatou, *Quel cette tatou, Ma quelle tatou (*Un which armadillo, Which this armadillo, My which armadillo*)

5.4 Inspecting Coverage and Correctness

In the previous sections, GRADE was used to generate MRSs and sentences *ex nihilo*. As mentioned above however, a core semantics can be used to restrict the set of output sentences to sentences whose MRS include this core semantics. This is useful for instance, to systematically inspect all variations output by the grammar on a given input. These variations include all morphological variations supported by the lexicon (number, tense, mode variations) and the syntactic variations supported by the grammar for the same MRSs (e.g., active/passive). It also includes the variations supported by GRADE in that some rules are not checked for semantic compatibility thereby allowing for additional materials to be added. In effect, GRADE allows for the inclusion of arbitrary determiners and auxiliaries.

Tree Family	MRS	Sent.	S/MRS
ilV	7	52	7.4
n0V	65	161	2.4
n0ClV	30	62	2.0
n0ClVn1	20	25	1.25
n0ClVden1	10	15	1.5
n0ClVpn1	40	63	1.57
n0Vn1	20	110	5.5
n0Van1	30	100	3.33
n0Vden1	5	15	3.00
n0Vpn1	25	76	3.04
ilVcs1	1	1	1.00
n0Vcs1	200	660	3.3
n0Vas1	35	120	3.42
n0Vn1Adj2	10	15	1.5
s0Vn1	4	24	6.00
n0Vn1n2	10	48	4.80
n0Vn1an2	5	45	9.00

Table 2: Producing Variants

Table 2 shows the number of MRSs and sentences output for each verb family given a matching core semantics and a morphological lexicon including verbs in all simple tenses (3rd person only) and nouns in singular and plural[6]. The ratio S/M of sentences on MRSs produced by one GRADE call shows how the underspecified core semantics permits exploring a larger number of sentences generated by the grammar than could be done by generating from fully specified MRSs. For the n0Vn1an2 class, for instance, the GRADE call permits generating 9 times more sentences in average than generating from a single MRS.

[5]Recall that in FB-LTAG, adjunction is the operation which permits applying recursive rules (i.e., auxiliary trees). Hence allowing for adjunctions amounts to allowing for modification with the exception already noted above of certain verbs subcategorising for sentential arguments.

[6]The lexicon used in this experiment includes more morphological variants than in the experiment of Section 5.2 where the focus was on syntactic rather than morphological variants. Hence the different number of generated sentences.

6 Conclusion

When using a grammar for generation, it is essential, not only that it has coverage (that it does not undergenerate) but also that it be precise (that it does not overgenerate). Nonetheless, relatively little work has been done on how to detect overgeneration. In this paper, we presented an algorithm and a methodology to explore the sentences generated by a grammar; we described an implementation of this algorithm based on DCGs (GRADE); and we illustrated its impact by applying it to an existing grammar. We showed that GRADE could be used to explore a grammar from different viewpoints: to find gaps or inconsistencies in the rule system; to systematically analyse the grammar account of functor/argument dependencies; to explore the interaction between base constructions and modifiers; and to verify the completeness and correctness of syntactic and morphological variants.

There are many directions in which to pursue this research. One issue is efficiency. Unsurprisingly, the computational complexity of GRADE is formidable. For the experiments reported here, runtimes are fair (a few seconds to a few minutes depending on how much output is required and on the size of the grammar and of the lexicon). As the complexity of the generated sentences and the size of the lexicons grow, however, it is clear that runtimes will become unpractical. We are currently using YAP Prolog tabling mechanism for storing intermediate results. It would be interesting however to compare this with the standard tabulating algorithms used for parsing and surface realisation.

Another interesting issue is that of the interaction between GRADE and error mining. As mentioned in Section 2, GRADE could be usefully complemented by error mining techniques as a means to automatically identify the most probable causes of errors highlighted by GRADE and thereby of improving the grammar. To support such an integration however, some means must be provided of sorting GRADE 's output into "good" and "bad" output i.e., into sentences that are grammatical and sentences that are over-generated by the grammar. We plan to investigate whether language models could be used to identify those sentences that are most probably incorrect. In a first step, simple and highly constrained input would be used to generate from the grammar and the lexicon a set of correct sentences using GRADE . Next these sentences would be used to train a language model which could be used to detect incorrect sentences produced by GRADE on more complex, less constrained input.

Other issues we are currently pursueing are the use of GRADE (i) for automating the creation of grammar exercises for learners of french and (ii) for creating a bank of MRSs to be used for the evaluation and comparison of data-to-text generators. The various degrees of under-specification supported by GRADE permit producing either many sentences out of few input (e.g., generate all basic clauses whose verb is of a given subcategorisation type as illustrated in Section 5.2); or fewer sentences out a more constrained input (e.g., producing all syntactic and morphological variants verbalising a given input semantics). We are currently exploring how semantically constrained GRADE calls permit producing variants of a given meaning; and how these variants can be used to automatically construct grammar exercises which illustrate the distinct syntactic and morphological configurations to be acquired by second language learners. In contrast, more underspecified GRADE calls can be used to automatically build a bank of semantic representations and their associated sentences which could form the basis for an evaluation of data-to-text surface realisers. The semantics input to GRADE are simplified representations of MRSs. During grammar traversal, GRADE reconstructs not only a sentence and its associated syntactic tree but also its full MRS. As a result, it is possible to produce a generation bank which, like the Redwook Bank, groups together MRSs and the sentences verbalising these MRSs. This bank however would reflect the linguistic coverage of the grammar rather than the linguistic constructions present in the corpus parsed to produce the MRS. It would thus provide an alternative way to test the linguistic coverage of existing surface realisers.

Acknowledgments

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

References

Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proc. of the 13th European Workshop on Natural Language Generation*.

E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, Ingria R., F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the DARPA Speech and Natural Language Workshop*, page 306311.

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

Ann Copestake, Alex Lascarides, and Dan Flickinger. 2001. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse, France.

Benoit Crabbé. 2005. *Reprsentation informatique de grammaires d'arbres fortement lexicalisées : le cas de la grammaire d'arbres adjoints.* Ph.D. thesis, Nancy University.

Daniël de Kok, Jianqiang Ma, and Gertjan van Noord. 2009. A generalized method for iterative error mining in parsing results. In *ACL2009 Workshop Grammar Engineering Across Frameworks (GEAF)*, Singapore.

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

Claire Gardent and Shashi Narayan. 2012. Error mining on dependency trees. In *Proceedings of ACL*.

Claire Gardent, Benjamin Gottesman, and Laura Perez-Beltrachini. 2010. Benchmarking surface realisers. In *COLING 2010 (Poster Session)*, Beijing, China.

Claire Gardent, Benjamin Gottesman, and Laura Perez-Beltrachini. 2011. Using regular tree grammar to enhance surface realisation. *Natural Language Engineerin*, 17:185–201. Special Issue on Finite State Methods and Models in Natural Language Processing.

Claire Gardent. 2008. Integrating a unification-based semantics in a large scale lexicalised tree adjoining grammar for french. In *COLING'08*, Manchester, UK.

Paul Klint, Ralf Lämmel, and Chris Verhoef. 2005. Toward an engineering discipline for grammarware. *ACM Transactions on Software Engineering Methodology*, 14(3):331–380.

Benoit Sagot and Eric de la Clergerie. 2006. Error mining in parsing results. In ACL, editor, *Proceedings of the ACL 2006*, pages 329–336, Morristown, NJ, USA.

Gertjan van Noord. 2004. Error mining for wide-coverage grammar engineering. In ACL, editor, *Proceedings of the ACL 2004*, pages 446–454, Morristown, NJ, USA.

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