Structure-Driven Lexicalist Generation

Shashi Narayan[1] Claire Gardent[2]
(1) Université de Lorraine/LORIA, Nancy, France
(2) CNRS/LORIA, Nancy, France
shashi.narayan@loria.fr, claire.gardent@loria.fr

Abstract
We present a novel algorithm for surface realisation with lexicalist grammars. In this algorithm, the structure of the input is used both top-down to constrain the selection of applicable rules and bottom-up to filter the initial search space associated with local input trees. In addition, parallelism is used to recursively pursue the realisation of each daughter node in the input tree. We evaluate the algorithm on the input data provided by the Generation Challenge Surface Realisation Task and show that it drastically reduce processing time when compared with a simpler, top-down driven, lexicalist approach.

Title and Abstract in Hindi

संरचना-प्रेरित शब्द-सज्जित उत्पादन

हम शब्द-सज्जित व्याकरण की मदद से वाक्यों के सतह संपादन के लिये एक नयी तकनीक प्रस्तुत कर रहे हैं। इस तकनीक में आगत संरचना का उपयोग दोनों “ऊपर से नीचे” उपयुक्त नियमों के चयन मे तथा “नीचे से ऊपर” आगत संरचना में मौजुद स्थानीय वृक्षों से जुड़े प्रांरभिक परीक्षण क्षेत्र को फिल्टर करने में किया गया है। इसके साथ, समांतरवाद का उपयोग आगत वृक्षों के शाखों की सतह संपादन हेतु कुशल तरीके से किया गया है। हम अपने तकनीक का मुल्यांकन “उत्पादन प्रतियोगिता: सतह संपादन” के आंकड़ों पर करते हैं तथा हम यह दिखातें हैं कि प्रस्तुत तरीका दूसरे साधारण “ऊपर से नीचे” प्रेरित शब्द-सज्जित तरीकों के मुकाबले उत्पादन समय को अत्यधिक घटा देता है।

Keywords: Generation, Tree Adjoining Grammar, Surface Realization.

Keywords in Hindi: उत्पादन, वृक्ष-सट-व्याकरण, सतह संपादन.

1 Introduction

Depending on the type of semantic representation encoded by the grammar, two main types of algorithms have been proposed for generating sentences with bi-directional, unification-based grammars such as CCG (Combinatory Categorial Grammar, (Espinosa et al., 2010)), HPSG (Head-Driven Phrase Structure Grammar, (Carroll et al., 1999)) and TAG (Tree Adjoining Grammar, (Gardent and Kow, 2005)).

For recursive semantic representations such as first-order logic formulae, head-driven algorithms (Shieber et al., 1990) have been argued to be best because they restrict the combinatorics inherent to bottom-up search; they avoid non termination by using lexical items to guide the search ; and they allow for semantically nonmonotonic grammars (i.e., grammars where the semantics of a rule at the left hand side need not be subsumed by the semantics of the rule at the right hand side). One main issue with this approach however is the so-called logical form equivalence problem (Shieber, 1993). A logic formula may have several logically equivalent but syntactically distinct formulae. For instance $p \wedge q$ is logically equivalent to $q \wedge p$. In general though, a grammar will associate with natural language expressions only one of these logically equivalent formula. Hence a generator will be able to produce the natural language expression E only when given the formula ϕ associated by the grammar with E. For all other formulae logically equivalent to ϕ, it will fail. Since, the problem of computing logical equivalence for first order logic is undecidable, the problem is quite deep.

For flat semantic representations such as MRSs (Minimal Recursion Semantics, (Copestake et al., 2001)) on the other hand, lexicalist approaches (Espinosa et al., 2010; Carroll and Oepen, 2005; Gardent and Kow, 2005) have extensively been used because (i) they impose few constraints on the grammar thereby making it easier to maintain bi-directional grammars that can be used both for parsing and for generation; and (ii) the approach eschews the logical form equivalence problem – Since the semantic representations are unstructured, there is no requirement on the generator to mirror a semantic structure. One known drawback of lexicalist approaches however is that they generally lack efficiency. Indeed, previous work has shown that the high combinatorics of lexicalist approaches stem from (i) strong lexical ambiguity (each input element is usually associated with a large number of grammatical structures thereby inducing a very large initial search space); (ii) the lack of order information in the input (as opposed to parsing where the order of words in the input string restricts the number of combinations to be explored); and (iii) intersective modifiers (given n modifiers applying to the same constituent, there are $n!$ ways to combine these together).

In this paper, we present an algorithm for surface realisation that combines techniques and ideas from the head-driven and the lexicalist approach. On the one hand, rule selection is guided, as in the lexicalist approach, by the elementary units present in the input rather than by its structure – In this way, the logical form equivalence issue is avoided. On the other hand, the structure of the input is used to provide top-down guidance for the search and thereby restrict the combinatorics.

To further improve efficiency, the algorithm integrates three additional optimisation techniques. From the lexicalist approach, it adapts two techniques designed to prune the search space, namely a so-called polarity filter on local input trees (Bonfante et al., 2004); and the use of a language model to prune competing intermediate substructures. In addition, the algorithm is parallelised to explore the possible completions of the top-down predictions simultaneously rather than sequentially.

The algorithm was implemented using a Feature-Based Lexicalised Tree Adjoining Grammar for English and tested on the Generation Challenge Surface Realisation task data (Belz et al., 2011). We compare our algorithm with a baseline lexicalist approach which processes the input tree top

down. The results show that the algorithm we propose drastically improves on the baseline, reducing generation time for sentences longer than 6 words w.r.t. this baseline.

This paper is structured as follows. Section 2 situates our approach with respect to related work. Section 3 introduces the input data provided by the Generation Challenge Surface Realisation task and used for the evaluation. Section 4 introduces the tree adjoining grammar used by the algorithm. Section 5 presents the surface realisation algorithm we developed. Section 6 describes the evaluation setup and the results obtained. Section 7 concludes with pointers for further research.

2 Related Work

Most of the recent proposals on optimising surface realisation with unification grammars focuses on lexicalist approaches, they place minimal requirements on the grammar and eschew the logical form equivalence problem. We now review the optimisation techniques used in these approaches. We also briefly review recent work on statistical approaches to surface realisation.

For HPSG, (Carroll and Oepen, 2005) present a bottom-up, lexicalist, surface realiser which uses a chart based strategy, subsumption-based local ambiguity factoring and a procedure to selectively unpack the generation forest according to a probability distribution given by a conditional, discriminative model. The algorithm is evaluated on the *hike* treebank, a collection of 330 sentences of instructional text taken from Norwegian tourism brochures with an average length of 12.8 words. Practical generation times average below or around one second for outputs of 15 words.

For TAG, (Gardent and Kow, 2007) propose a three step surface realisation algorithm for FB-LTAG (Feature-Based Lexicalised Tree-Adjoining Grammar) where first, a so-called polarity filter is used to prune the initial search space second, substitution is applied to combine trees together and third, adjunction is applied.

In essence, polarity filtering filters out combinations of FB-LTAG elementary trees which cover the input semantics but cannot yield a valid parse tree either because a syntactic requirement cannot be satisfied or because a syntactic resource cannot be used. In this way, the exponential impact of lexical ambiguity can be reduced. Furthermore applying substitution before adjunction means that first a skeleton sentence is built before modifiers are adjoined. This permits reducing the combinatorics introduced by intersective modifiers as the multiple intermediate structures they may license do not propagate to the rest of the sentence tree. In practice however, evaluation is restricted to short input and the algorithm fails to scale up (Gardent and Perez-Beltrachini, 2010).

(Koller and Striegnitz, 2002) present a surface realisation algorithm where (i) the XTAG FB-LTAG grammar (The XTAG Research Group, 2001) is converted to a dependency grammar capturing the derivation trees of XTAG and (ii) a constraint-based dependency parser is used to construct derivation trees from semantic representations. The parser used was specifically developed for the efficient parsing of free word order languages and is shown to efficiently handle both the lexical ambiguity and the lack of order information in the input that are characteristic of surface realisation from a flat semantics. The evaluation however is restricted to a few hand constructed example inputs; and the grammar conversion ignores feature structure information.

To address these shortcomings, (Gardent and Perez-Beltrachini, 2010) present an approach which makes use of the procedure for converting an FB-LTAG to a Feature-Based Regular Tree Grammar (FB-RTG) described in (Schmitz and Roux, 2008). Like in (Koller and Striegnitz, 2002), the initial FB-LTAG is converted to a grammar of its derivation trees. However in this case, the grammar conversion and the resulting feature-based RTGs accurately translates the full range of unification

mechanisms employed in the initial FB-LTAG. An Earley, bottom-up algorithm is developed and the approach is tested on a large benchmark of artificially constructed examples illustrating different levels of linguistic complexity (different input lengths, different numbers of clauses and of modifiers). The approach is shown to outperform the algorithm presented in (Gardent and Kow, 2007) in terms of space. Speed is not evaluated however and the algorithm is not evaluated on the real life data.

Probabilistic techniques have also been proposed to improve e.g., lexical selection, the handling of intersective modifiers and the selection of the best output. For instance, (Bangalore and Rambow, 2000) uses a tree model to produce a single most probable lexical selection while in CCG based White's system (White, 2004), the best paraphrase is determined on the basis of n-gram scores. To address the fact that there are $n!$ ways to combine any n modifiers with a single constituent, (White, 2004) proposes to use a language model to prune the chart of identical edges representing different modifier permutations, e.g., to choose between *fierce black cat* and *black fierce cat*. Similarly, (Bangalore and Rambow, 2000) assumes a single derivation tree that encodes a word lattice (*a {fierce black, black fierce} cat*), and uses statistical knowledge to select the best linearisation. Recently, (Espinosa et al., 2008) adapted the supertagging techniques first proposed for parsing (Bangalore and Joshi, 1999) to surface realisation. Given a treebank in the appropriate format, this technique permits filtering the initial search space by using a model trained on that treebank. Supertagging was shown to improve the performance of symbolic parsers and generators significantly. However, it requires the existence of a treebank in a format appropriate to generate the supertagging model.

In sum, various symbolic and statistical techniques have been developed to improve the efficiency of grammar-based surface realisation. However, statistical systems using supertagging require the existence of a treebank in an appropriate format while the purely symbolic systems described in (Carroll and Oepen, 2005; Gardent and Kow, 2005; Koller and Striegnitz, 2002; Gardent and Perez-Beltrachini, 2010) have not been evaluated on large corpora of arbitrarily long sentences such as provided by the surface realisation (SR) task (Belz et al., 2011).

Recently, (Guo et al., 2011; Bohnet et al., 2011; Stent, 2011) have developed statistical dependency realisers that do not make use of an explicit grammar but use cascaded classifiers and n-gram models to map in SR input data to sentences. They obtain the best results in the SR task partly because, for grammar based systems, converting the provided input into the format expected by the grammar proved to be extremely difficult.

The algorithm we propose departs from these approaches in that it is a grammar-based approach; it is optimised by combining parallel processing, top-down prediction and local bottom-up polarity filtering; and it was evaluated on a large scale using the input data provided by Generation Challenge SR Task.

3 Input Representations

Recently, the Generation Challenge has promoted a Surface Realisation (SR) task (Belz et al., 2011) where the input provided to test and compare surface realisers are (deep or shallow) dependency structures. Here we assume as input to surface realisation, the shallow dependency structures provided by this task namely, unordered trees whose edges are labelled with syntactic functions and whose nodes are labelled with lemmas, part of speech tags, partial morphosyntactic information such as tense and number and, in some cases, a sense tag identifier. All words of the original sentence are represented by a node in the tree. An example of the shallow dependency trees used

for the SR task is given in Figure 1.

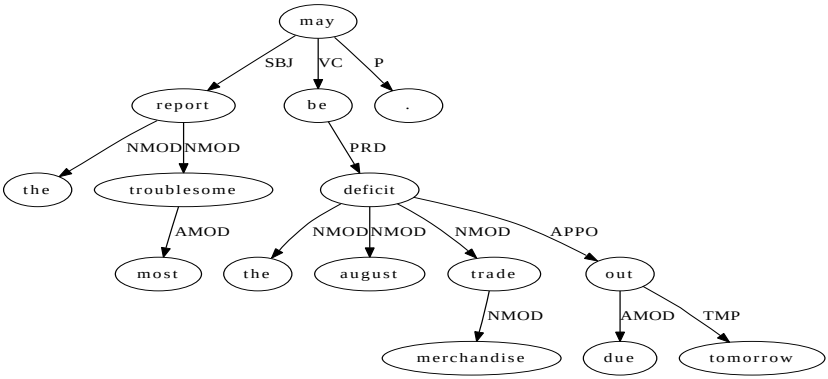


Figure 1: Input shallow dependency tree from the Generation Challenge Surface Realisation Task for the sentence "*The most troublesome report may be the August merchandise trade deficit due out tomorrow .*"

Note that contrary to the flat semantic representations often used by surface realisers, the SR data has a clear tree structure. Thus the combinatorics induced by the lack of order in flat semantic representations is less in this task. Indeed, the algorithm we present exploits this structure to minimize the combinatorics. Similarly, (White, 2006) applies chunking constraints to the graph structure of flat semantic representation to constrain the generation of coordinate structures and address the issue of semantically incomplete phrases.

4 Grammar

Following (Gardent and Perez-Beltrachini, 2010), we perform surface realisation using a Feature-Based Regular Tree Grammar (FB-RTG) describing the derivation trees of a Feature-Based Lexicalised Tree Adjoining Grammar (FB-LTAG, (Joshi and Schabes, 1996)) rather than the FB-LTAG itself. In what follows, we briefly introduce FB-LTAG and the derived FB-RTG used for generation.

4.1 FB-LTAG

The grammar underlying the surface realisation algorithm presented in the next section is an FB-LTAG for English consisting of roughly 1000 trees and whose coverage is similar to XTAG (The XTAG Research Group, 2001).

Figure 2 shows an example FB-LTAG. Briefly, an FB-LTAG consists of a set of elementary trees which can be either initial or auxiliary. Initial trees are trees whose leaves are labeled with substitution nodes (marked with a down-arrow) 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. In addition, in an FB-LTAG, each elementary tree is anchored by a lexical item (lexicalisation) and the nodes in the elementary trees are decorated with two feature structures called *top* and *bottom* which are unified during derivation. 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. Derivation in an FB-LTAG yields two trees: a *derived tree* which is, like for context free grammars, the tree produced by combining the grammar rules (here, the elementary trees) licensed by the input; and a *derivation tree* which indicates how

the derived tree was built i.e., which elementary trees were used and how they were combined. Figure 3 show the derived and derivation trees associated by the grammar shown in Figure 2 with the sentence "*Which fruit has John eaten?*". For a detailed presentation of the FB-LTAG formalism, the reader is referred to (Vijay-Shanker and Joshi, 1988).

4.2 FB-RTG

As shown in (Koller and Striegnitz, 2002; Gardent and Perez-Beltrachini, 2010), processing the derivation trees of a given FB-LTAG rather than its derived trees is more efficient. Following (Gardent and Perez-Beltrachini, 2010), we therefore use not the initial FB-LTAG described in the previous section, but the FB-RTG grammar of derivation trees that can be derived from it. That is, the surface realisation algorithm first builds a derivation tree. The generated sentence is then extracted from the derived tree[1] which can be reconstructed from this derivation tree using the original FB-LTAG.

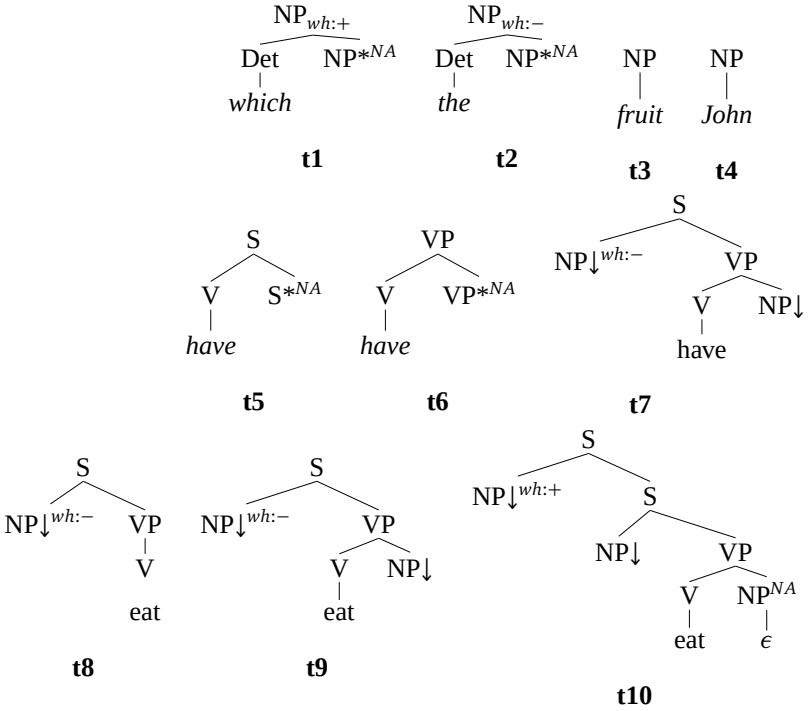
Figure 2 shows an example FB-LTAG and the corresponding FB-RTG. The conversion from FB-LTAG to FB-RTG is described in detail in (Schmitz and Roux, 2008). Intuitively, the FB-RTG representation of an FB-LTAG elementary tree t, is a rule whose left hand side (LHS) describes the syntactic requirement satisfied by t (e.g., S_S for an initial tree rooted in S and VP_A for an auxiliary tree rooted in VP) and whose right hand side (RHS) describes its requirements. Adjunction is handled as an optional requirement which can be satisfied by the adjunction of an empty string and subscripts indicates the nature of the requirement (S for a substitution and A for adjunction). For instance, the rule **r8** in Figure 2 repeated below for convenience, describes the contribution of the elementary tree **t8** lexicalised with the lemma *eat* to a derivation tree as follows: **t8** can satisfy a requirement for a substitution on a node labelled with the S category (LHS with the category S_S) and requires one substitution on a node labelled with the NP category (NP_S on the RHS) and two optional adjunctions of category S and VP respectively (S_A, VP_A on the RHS).

$$S_S^{[t:T,b:B]} \rightarrow eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A)$$

The derivation process in FB-RTG produces trees that are almost identical to the FB-LTAG derivation trees. Figure 3 shows the FB-LTAG derived, FB-LTAG derivation and FB-RTG derived tree for the sentence "*Which fruit has John eaten?*". When abstracting away from the categorial nodes, the FB-RTG derivation tree mirrors the derivation tree of the original FB-LTAG. The A and S subscripts indicate which operation was used for combining; and the nodes at which each FB-LTAG elementary tree adjoins or substitutes is encoded by features in these trees: for instance, the subject node of **t9** will have the feature *subject* while its object node will have the feature *object*. By comparing the dependency relations present in the input tree with the feature values given by the grammar, it is thus possible to determine on which nodes of the mother tree in the derivation tree, its daughter trees should combine.

Note that the FB-RTG tree is unordered. During generation, the appropriate linearisation of the lexical items is obtained by constructing the FB-LTAG derived tree from the FB-RTG derivation tree. Morphological realisation is carried out in a post-processing step from the list of lemmas and feature structures decorating the yield of the FB-LTAG derived tree.

[1] in FB-LTAG, the mapping from derivation tree to derived tree is one-to-one.



r1	$NP_A^{[t:T]}$	$\rightarrow$	$\mathit{which}(NP_A^{[t:T,b:[wh:+]]})$
r2	$NP_A^{[t:T]}$	$\rightarrow$	$\mathit{the}(NP_A^{[t:T,b:[wh:-]]})$
r3	$NP_S^{[t:T]}$	$\rightarrow$	$\mathit{fruit}(NP_A^{[t:T]})$
r4	$NP_S^{[t:T]}$	$\rightarrow$	$\mathit{John}(NP_A^{[t:T]})$
r5	$S_A^{[t:T]}$	$\rightarrow$	$\mathit{have}(S_A^{[t:T]})$
r6	$VP_A^{[t:T]}$	$\rightarrow$	$\mathit{have}(VP_A^{[t:T]})$
r7	$S_S^{[t:T,b:B]}$	$\rightarrow$	$\mathit{have}(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S)$
r8	$S_S^{[t:T,b:B]}$	$\rightarrow$	$\mathit{eat}(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A)$
r9	$S_S^{[t:T,b:B]}$	$\rightarrow$	$\mathit{eat}(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S)$
r10	$S_S^{[t:T,b:B]}$	$\rightarrow$	$\mathit{eat}(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A)$
r11	$X_A^{[t:T,b:T]}$	$\rightarrow$	ϵ

Figure 2: A toy FB-LTAG and the corresponding FB-RTG. For the sake of clarity, feature structures are abbreviated. **r11** (not present in the original FB-LTAG) implements optional adjunction for arbitrary categories with X, a variable ranging over all syntactic categories.

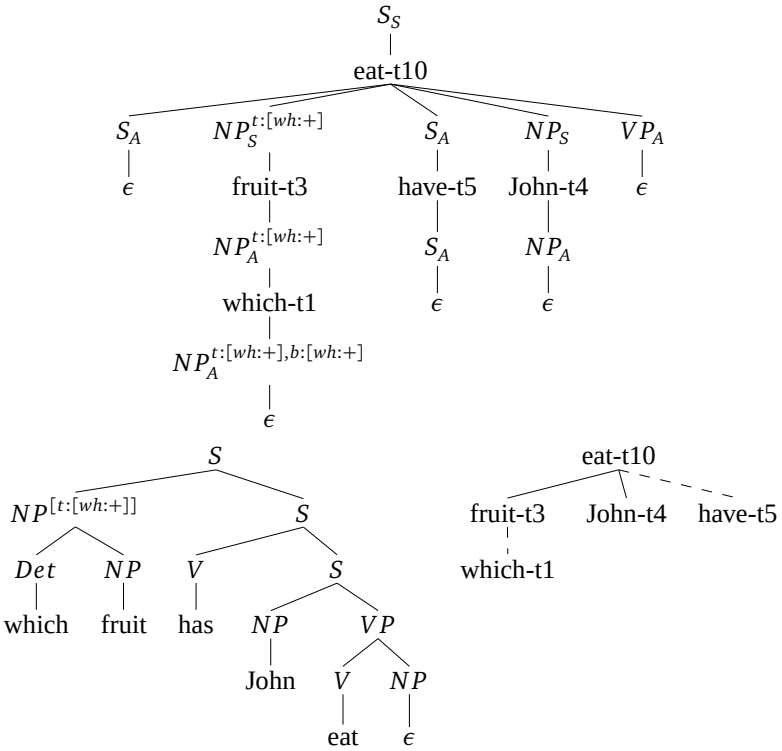


Figure 3: The FB-RTG derivation for "*Which fruit has John eaten*" and the corresponding FB-LTAG derived and derivation trees. In the derivation tree, the nodes are labelled with a lemma/FB-LTAG tree name pair; dashed lines indicate adjunction and solid lines substitution. Adjunction and substitution sites have been omitted.

5 The Surface Realisation Algorithm

Surface realisation starts from the root node of the input tree and processes all children nodes in parallel by spreading the lexical selection constraints top-down and completing the FB-RTG rules bottom-up. Figure 4 shows the architecture of the surface realiser. The controller provides the interface to our surface realization system. It takes a shallow dependency tree as input and produces a ranked list of sentences as output. More specifically, the controller defines a process pool such that each process present in this pool represents a node (a lemma) in the input dependency tree and the communication scheme among processes reflects the dependency relations among nodes in the input dependency tree. In this way, generation is guided by the structure of the input dependency tree.

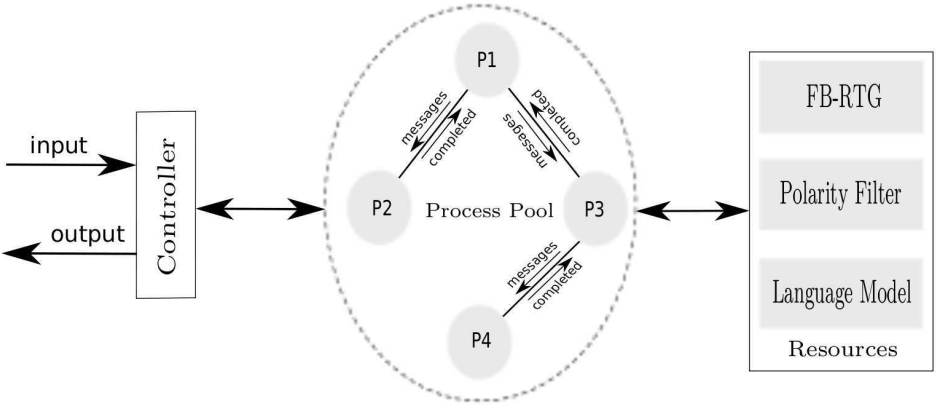


Figure 4: A Parallel Architecture for Surface Realisation

The algorithm proceeds in five major steps as follows.

Top-Down Rule selection and Filtering. Starting from the root node, the input dependency tree is traversed top-down to associate each node in the input tree with a set of grammar rules (from the FB-RTG). This step corresponds to the lexical lookup phase of lexicalist approaches whereby each literal in the input selects the grammar rules whose semantics subsumes this literal. Our approach differs from existing lexicalist approaches however in that it uses the top-down information given by the structure of the input to filter out some possibilities that cannot possibly lead to a valid output. More precisely, for each input node *n* with lemma *w*, only those rules are selected which are associated with *w* in the lexicon. In addition, the left-hand side (LHS) category of each selected rule must occur at least once in the right-hand side (RHS) of the rules selected by the parent node.

For instance, given the input dependency tree shown in Figure 5 for the sentence "*Which fruit has John eaten?*", and the grammar given in Figure 2, all rules **r8**, **r9** and **r10** associated with the lemma '*eat*' will be selected because all of them corresponds to the S[2] rooted initial trees[3].

[2]The controller triggers the root process "eat" with the initial lexical selection constraint (S_S, S rooted initial trees) to generate complete sentences.

[3]The grammar is lexicalised with lemmas rather than forms. The appropriate forms are generated at the end of the generation process based on the lemmas and on the feature structures decorating the yield of the trees output by the generator.

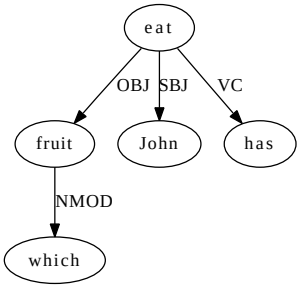


Figure 5: Dependency Tree for "*Which fruit has John eaten*"

$$\begin{array}{llllll} \surd & \textbf{r8} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A) \\ \surd & \textbf{r9} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\ \surd & \textbf{r10} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A) \end{array}$$

The parent process creates a new lexical selection constraint message consisting of its RHS requirements in selected RTG rules and passes it to its children processes. In Figure 5, the process associated with the node '*eat*' will send a message consisting of S_A, NP_S and VP_A (RHS requirements of rules **r8**, **r9** and **r10**) to its children processes associated with '*fruit*', '*John*' and '*have*'.

Starting from the trigger initiated by the *controller*, the process of message spreading happens in recursive and parallel manner throughout the process pool reflecting the input dependency tree in a top-down fashion. It eliminates all RTG rules which cannot possibly lead to a valid output well before carrying out any substitution and adjoining operation on the RTG rules.

For instance, the rule **r7** for '*have*' will not be selected because its left-hand side is S_S which does not satisfy the lexical selection constraints (S_A, NP_S and VP_A) sent by its parent '*eat*'.

$$\begin{array}{llllll} \surd & \textbf{r5} & S_A^{[t:T]} & \rightarrow & have(S_A^{[t:T]}) \\ \surd & \textbf{r6} & VP_A^{[t:T]} & \rightarrow & have(VP_A^{[t:T]}) \\ \times & \textbf{r7} & S_S^{[t:T,b:B]} & \rightarrow & have(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \end{array}$$

Leaf closure. When reaching the leaf nodes of the input tree, the top and bottom feature structures of the rules selected by these leaf nodes are unified. The completed rules of a leaf node are sent back to its parent.

Local Polarity filtering. As mentioned in Section 2, polarity filtering (Gardent and Kow, 2005) eliminates from the search space those sets of rules which cover the input but cannot possibly lead to a valid derivation either because a substitution node cannot be filled or because a root node fails to have a matching substitution site[4] While (Gardent and Kow, 2005) applies polarity filtering to the initial search space (the set of rules selected by all literals in the input), we apply polarity filtering to each local tree while going up the input tree. Thus, this filtering will weed out all combinations

[4]Since it only eliminates combinations that cannot possibly lead to a valid parse, polarity filtering does not affect completeness. Nor does it place any particular constraint in the grammar. All that is required is that the grammar encodes a notion of resources and requirements i.e., of items that cancel each other out. Typically, grammar rules support this constraint in that e.g., the left-hand side of a rule and one category in the right-hand side of another rule can be viewed as canceling each other out if they match.

of mother rules and completed immediate daughter rules which cannot possibly yield a complete tree either because some daughter rule cannot be used or because some requirement of the mother rule cannot be satisfied. For instance, after processing the daughters of the '*eat*' node in the input dependency tree shown in Figure 5, all combinations of **r8** (intransitive '*eat*') with the daughter trees will be excluded. This is because at this stage of processing, the trees built bottom up for '*which fruit*', '*John*' and '*have*' includes two NPs with LHS category NP_S (Figure 6) while the **r8** rule only requires one such NP. That is, for this rule, the completed daughter rule for *which fruit* will show up as a superfluous syntactic resource.

$$\begin{array}{llllll} \times & \textbf{r8} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A) \\ \surd & \textbf{r9} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\ \surd & \textbf{r10} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A) \end{array}$$

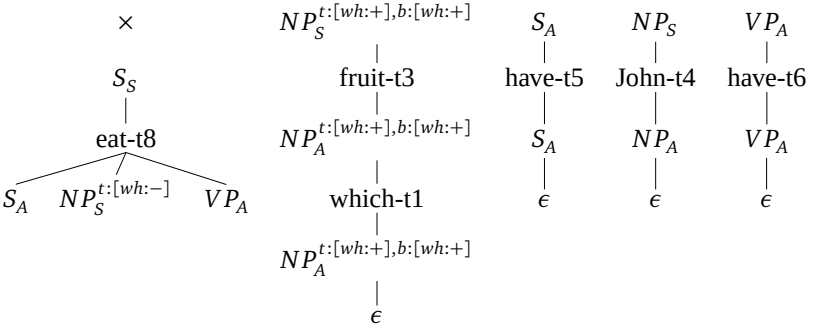


Figure 6: Polarity Filtering will filter out the **r8** rule for '*eat*' since one of trees '*which fruit*' or '*John*' would then appear as a superfluous syntactic resource as illustrated by the above derivation.

By restricting polarity filtering to local input trees, we avoid the computation of the very large automaton required when filtering the global initial search space as done in (Gardent and Kow, 2005).

As noted by one of our reviewers, supertagging models can probably approximate local polarity filtering. For instance, a supertagger model might learn that an intransitive category is very unlikely whenever the input dependency tree contains one or more core arguments.

The combined effect of top-down filtering and local polarity filtering avoids considering most of RTG rules which can never lead to valid output well before carrying out any substitution and adjoining operation on the RTG rules to try to complete them. The Earley, bottom-up algorithm (Gardent and Perez-Beltrachini, 2010) also achieves some amount of top-down filtering during its prediction stage but the lexical selection constraint is limited to the top of the RHS requirements of the RTG rule being processed, hence it may try completing the RTG rules which cannot possibly lead to a valid output whereas in our proposed approach all RHS requirements of the selected RTG rules are available as the lexical selection constraint information during both top-down filtering and local polarity filtering steps.

Bottom-Up generation. For each local tree in the input, the rule sets passing the local polarity filter are tried out for combination. The completed daughter RTG rules are combined to the local

initialized RTG rule using substitution and adjoining operations. The local initialized RTG rule fails to complete if any feature conflicts are found.

Note that for each rule set let through by polarity filtering, the category and the number of daughter trees exactly match the requirement of the associated mother rule. For instance, as explained above, the rule **r8** representing an intransitive use of the verb '*eat*' is ruled out by polarity filtering since it does not permit "consuming" the NP_S resource provided by one of NPs '*which fruit*' or '*John*'. Conversely, given an input tree of the form `eat(john, has)`, the rules **r9** and **r10** representing a transitive use of the verb '*eat*' would be filtered out by polarity filtering. As a result, the intermediate structure shown below will not be computed. That is, while the global polarity filtering used in (Gardent and Kow, 2005) permits weeding out global combination of trees that are invalid, local polarity filtering additionally permits reducing the number of intermediate structures built first, because there is no need for prediction i.e., for active chart items and second, because intermediate structures that cannot possibly lead to a valid derivation are not built.

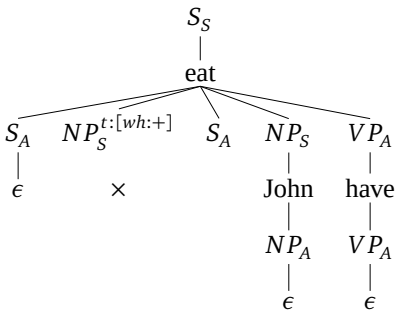


Figure 7: Given the input tree `eat(john, has)`, local polarity filtering filters out this intermediate structure because it cannot be completed given the input

N-gram filtering using a Language Model. To further prune the search space and to appropriately handle word order, the SR algorithm also integrates a language model and can be parametrized for the number of best scoring n-grams let through after each bottom-up generation step. In this way, not all possible orderings of intersective modifiers are produced, only those that are most probable according to the language model.

6 Empirical Evaluation

We now report on the results obtained when running the algorithm and the grammar described above on the shallow input data provided by the Generation Challenge Surface Realisation Task. Because we are presenting an algorithm for surface realisation rather than a surface realiser, the main focus of the evaluation is on speed (not coverage or accuracy). Nevertheless, we also report coverage and BLEU score as an indication of the capabilities of the surface realiser i.e., the algorithm combined with the grammar and the lexicon.

6.1 Runtimes

The SR data on which we evaluate our surface realisation algorithm are the shallow dependency trees described in Section 3.

We use as a baseline the FB-RTG based lexicalist approach (**B**ASELINE) described in (Narayan, 2011). In this approach, FB-RTG rules are selected top-down following the structure of the input dependency tree and all FB-RTG rules selected for a given local input tree are tried out for combination using a chart-based approach. This baseline thus permits observing the impact of the various optimisations described below. In future work, it would be interesting to obtain time information from the systems participating in the SR challenge and to compare them with those of our system.

TDBU-PAR (top-down, bottom-up and parallelised) is the algorithm presented here running on a 4 core system. To evaluate the impact of parallelism on runtimes, we also computed runtimes for a sequential version of the same algorithm (**T**DBU-SEQ). In **T**DBU-SEQ, daughter subtrees (processes) of the input dependency tree are processed sequentially.

Table 1 shows the runtimes for the three surface realisation algorithms **B**ASELINE, **T**DBU-SEQ and **T**DBU-PAR with varying sizes of sentences. For the TDBU algorithms, the n-gram filtering is set to 10 that is, for each local input tree, the 10 best n-grams are passed on. We split the data into 4 sets according to the input length where the input length is the number of nodes (or words) in the input dependency tree. The average number of words in a sentence in the first set $S(0-5)$ is 4, in the second set $S(6-10)$, 7, in the third set $S(11-20)$, 15, and in the final set $S(All)$ (all lengths), 17. The maximum length of a sentence in the final set $S(All)$ is 74. To make comparisons between **B**ASELINE, **T**DBU-SEQ and **T**DBU-PAR possible, the maximum arity of words present in the input dependency trees is set to 3 (because BASELINE mostly fails on input containing nodes with higher arity).

Algorithm	Sentences (Length L)							
	$S(0-5)$		$S(6-10)$		$S(11-20)$		$S(All)$	
	Total	Succ	Total	Succ	Total	Succ	Total	Succ
	1084	985	2232	1477	5705	520	13661	2744
BASELINE	0.85	0.87	10.90	10.76	110.07	97.52	–	–
TDBU-SEQ	1.49	1.63	2.84	3.64	4.36	6.03	4.52	3.18
TDBU-PAR	1.53	1.66	2.56	3.28	2.66	4.14	2.57	2.78

Table 1: Comparison between generation times (seconds)

BASELINE turns out to be faster than **T**DBU-PAR and **T**DBU-SEQ for sentences of smaller length (≤ 5). It can be explained because of the parallelism and the multiprocessing overhead. But **T**DBU-PAR and **T**DBU-SEQ leaves behind **B**ASELINE for longer sentences. For input longer than 10, the simple **B**ASELINE algorithm times out whereas **T**DBU-PAR remains stable. For $S(All)$, **T**DBU-PAR achieves a reasonable average of 2.57 seconds for all sentences (Total) and 2.78 seconds for successful sentences (Succ).

Table 1 does not show a big difference in performance between **T**DBU-PAR and **T**DBU-SEQ because the maximum arity of the input dependency trees is kept low (maximum 3). In Table 2, we split the data by arity whereby the dataset $S(i)$ consists of input dependency trees with maximum arity i. As can be seen, the difference between the two algorithms steadily increases with the arity of the input thereby demonstrating the impact of parallelism.

6.2 Coverage and Accuracy

The grammar and lexicon used to test the surface realisation algorithm presented in this paper are under development so that coverage and accuracy are still low. Table 3 shows the coverage and

Algorithm	Sentences (Arity)											
	$S(1)$		$S(2)$		$S(3)$		$S(4)$		$S(5)$		$S(6)$	
	Total	Succ	Total	Succ	Total	Succ	Total	Succ	Total	Succ	Total	Succ
	190	178	1218	964	3619	1039	5320	605	2910	137	1093	18
TDBU-SEQ	0.89	0.94	2.52	2.63	3.65	3.39	5.07	4.54	5.24	4.62	8.20	7.29
TDBU-PAR	0.97	1.03	2.35	2.50	2.63	3.10	2.91	3.77	2.86	3.88	3.09	4.76

Table 2: Comparison between generation times (seconds) with varying arities.

accuracy (on the covered sentences) results obtained for sentences of size 6 (S-6), 8 (S-8) and all (S-All). The dataset S-All differs from the dataset $S(All)$ discussed in previous section. S-All considers all sentences without any restriction over the maximum arity in the input dependency trees. S-All consists of 26725 sentences with the average length of 22 and the maximum length of 134. The maximum arity in these sentences varies from 1 to 18 with an average of 4.

Data Type	Total	Coverage (#)		Coverage (%)	BLEU Score
		Covered	Uncovered		
S-6	3877	3506	371	90.43	0.835
S-8	3583	3038	545	84.79	0.800
S-All	26725	10351	16374	38.73	0.675

Table 3: Coverage and Bleu Scores for covered sentences.

As can be seen coverage markedly decreases for longer sentences. Error mining on this data indicates that failure to generate is due mostly to complex sentence coordinations (e.g., verb coordination, gapping phenomenon) (Sarkar and Joshi, 1996) which could be very common in sentences of average length 22 in S-All. Other failure causes are inadequate treatments of multiword expressions and foreign words.

7 Conclusion

We presented a novel algorithm for surface realisation with lexicalised grammar which takes advantage of the input structure (a tree) to filter the initial search space both top-down and bottom up; and to parallelise processes. We evaluated this algorithm on large scale data and showed that it drastically reduces runtimes on this data when compared to a simple lexicalist approach which explores the whole search space.

As mentioned in section 3, the input data provided by the SR task differs from the flat semantic representations assumed by most existing surface realisers in that it displays a clear tree structure. The algorithm presented here makes use of that structure to optimize performance. In future work, we plan to investigate whether the hybrid top-down, bottom-up approach we developed to guide the SR search can be generalised to the graph structure of semantic representations.

Acknowledgments We would like to thank Laura Perez-Beltrachini for useful discussion on generation using the FB-RTG. We are grateful to the Generation Challenge SR Task organisers for providing us with the SR data and to Anja Belz and Mike White for providing us with the task evaluation scripts. The research presented in this paper was partially supported by the European Fund for Regional Development within the framework of the INTERREG IV A Allegro Project.

References

Bangalore, S. and Joshi, A. K. (1999). Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Bangalore, S. and Rambow, O. (2000). Using TAGs, a tree model and a language model for generation. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, Paris, France.

Belz, A., White, M., Espinosa, D., Kow, E., Hogan, D., and Stent, A. (2011). The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the 13th European Workshop on Natural Language Generation (ENLG)*, Nancy, France.

Bohnet, B., Mille, S., Favre, B., and Wanner, L. (2011). <stumaba >: From deep representation to surface. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 232–235, Nancy, France. Association for Computational Linguistics.

Bonfante, G., Guillaume, B., and Perrier, G. (2004). Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, Geneva, Switzerland.

Carroll, J., Copestake, A., Flickinger, D., and Paznański, V. (1999). An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 86–95, Toulouse, France.

Carroll, J. and Oepen, S. (2005). High efficiency realization for a wide-coverage unification grammar. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP)*, Jeju Island, Korea.

Copestake, A., Lascarides, A., and Flickinger, D. (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.

Espinosa, D., Rajkumar, R., White, M., and Berleant, S. (2010). Further meta-evaluation of broad-coverage surface realization. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 564–574, Cambridge, MA. Association for Computational Linguistics.

Espinosa, D., White, M., and Mehay, D. (2008). Hypertagging: Supertagging for surface realization with CCG. In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL): Human Language Technologies (HLT)*, pages 183–191, Columbus, Ohio. Association for Computational Linguistics.

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

Gardent, C. and Kow, E. (2007). A symbolic approach to near-deterministic surface realisation using tree adjoining grammar. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 328–335, Prague, Czech Republic. Association for Computational Linguistics.

Gardent, C. and Perez-Beltrachini, L. (2010). RTG based surface realisation for TAG. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 367–375, Beijing, China.

Guo, Y., Hogan, D., and van Genabith, J. (2011). Dcu at generation challenges 2011 surface realisation track. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 227–229, Nancy, France. Association for Computational Linguistics.

Joshi, A. K. and Schabes, Y. (1996). Tree-adjoining grammars. *Handbook of Formal Languages and Automata*.

Koller, A. and Striegnitz, K. (2002). Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL)*, Philadelphia.

Narayan, S. (2011). RTG based surface realisation from dependency representations. Master's thesis, University of Nancy 2 and University of Malta. Erasmus Mundus Master "Language and Communication Technology".

Sarkar, A. and Joshi, A. K. (1996). Coordination in tree adjoining grammars: Formalization and implementation. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 610–615.

Schmitz, S. and Roux, J. L. (2008). Feature unification in TAG derivation trees. In *Proceedings of the 9th International Workshop on Tree Adjoining Grammars and Related Formalisms*, Tubingen, Germany.

Shieber, S. (1993). The problem of logical form equivalence. *Computational Linguistics*, 19(1):179–190.

Shieber, S. M., Noord, G. V., Pereira, F. C. N., and Moore, R. C. (1990). Semantic-head-driven generation. *Computational Linguistics*, 16.

Stent, A. (2011). Att-0: Submission to generation challenges 2011 surface realization shared task. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 230–231, Nancy, France. Association for Computational Linguistics.

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

Vijay-Shanker, K. and Joshi, A. (1988). Feature structures based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary.

White, M. (2004). Reining in CCG chart realization. In *Proceedings of the third International Conference on Natural Language Generation (INLG)*, pages 182–191, Brighton, UK.

White, M. (2006). Efficient realization of coordinate structures in combinatory categorial grammar. *Research on Language and Computation*, 4(1):39–75.