Syntactic Generation

Gertjan van Noord
BCN Groningen

June 13, 1994

1 Syntactic Generation

1.1 Introduction

We can often distinguish two components in a natural language generation module. On the one hand it should be decided *what* should be said. This task is delegated to a 'planning' component. Such a component might produce an expression representing the content of the proposed utterance. On the basis of this representation the syntactic generation component produces the actual output sentence(s). Although the distinction between planning and syntactic generation is not uncontroversial, we will nonetheless assume such an archictecture here, in order to explain some of the issues that come up in syntactic generation.

A (natural language) grammar is a formal device that defines a relation between (natural language) utterances and their corresponding meanings. In practice this usually means that a grammar defines a relation between strings and logical forms. During natural language understanding, the trick is to come up with the logical form that corresponds to the input string. Syntactic generation can be described as the problem to find the corresponding string for an input logical form.

We are thus making a distinction between the grammar which *defines* this relation, and the procedure that *computes* the relation on the basis of such a grammar. In the current state of the art unification-based (or more general: constraint-based) formalisms are used to express such grammars. The actual format of the grammar formalism does not matter much here; many issues in syntactic generation can be explained by assuming a very simple variant from the class of constraint-based formalisms, such as DCG.

It is clearly an interesting objective to use one and the same grammar for both natural language understanding and generation. In fact the potential for reversibility was a primary motivation for the introduction of Martin Kay's Functional Unfication Grammar. In recent years interest in such a *reversible* architecture has led to a number of publications. [1]

Almost any modern linguistic theory assumes, that a natural language grammar not only describe the correct sentences of a language, but that such a grammar also describes the corresponding semantic structures of the grammatical sentences. Given that a grammar specifies the relation between phonology and semantics it seems obvious that the generator is supposed to use this specification. For example, for Generalized Phrase Structure Grammars (GPSG) [9][chapters 9 and 10] provide a detailed description of the semantic interpretation of the sentences licensed by the grammar. Thus one might assume that a generator based on GPSG constructs a sentence for a given semantic structure, according to the semantic interpretation rules of GPSG. Alternatively, [4] presents a generator, based on GPSG, which does not take as its input a logical form, but rather some kind of control expression which merely instructs the grammatical component which rules of the grammar to apply. Similarly, in the conception of [8], a generator is provided with some kind of 'deep structure' which can be interpreted as a control expression instructing the grammar which

[1] See for example [18] which is a collection of papers based on the 1991 ACL workshop 'Reversible Grammar in Natural Language Processing'; some other references are [2, 11, 7, 17, 6, 20].

rules to apply. These approaches to the generation problem clearly 'solve' some of the problems encountered in generation — simply by pushing the problem away into the conceptual component. In this overview I restrict the attention to the more ambitious approach sketched above.

1.2 Generation Techniques

An example. To illustrate the interest there is for various generation techniques we present a very simple, naive generation procedure. It is assumed that the grammar is represented as a simple DCG in which non-terminals consists of two parts: the part describing the syntactic properties of the phrase, and the part describing the logical form associated with that phrase. The two parts are distinguished by a slash. A small grammar constructing very simple 'logical forms' might be defined as:

```
 (1) s/Sem --> np/Subj, vp(Subj)/Sem.
 (2) vp(A)/left(A) --> [left].
 (3) vp(A)/kissed(A,Obj) --> [kissed], np/Obj.
 (4) vp(A)/kissed(Obj,A) --> [was], [kissed], pp(_)/by(_,A).
 (5) np/john --> [john].
 (6) np/mary --> [mary].
 (7) np/Sem --> [the], n/Sem.
 (8) n/Sem --> n/Sub, pp(Sub)/Sem.
 (9) n/man --> [man].
(10) n/woman --> [woman].
(11) pp(E)/by(E,Sem) --> [by], np/Sem.
(12) pp(E)/with(E,Sem) --> [with], np/Sem.
```

A very simple generation strategy would be the following. We start with the start symbol of the grammar, associated with the logical form from which we wish to generate. For example:

```
s/kissed(john,mary)
```

This can be rewritten using rule (1) into the sequence

```
< np/Subj, vp(Subj)/kissed(john,mary) >
```

The first element of this sequence can be rewritten as follows, using rule (6):

```
< [mary], vp(mary)/kissed(john,mary) >
```

The remaining non-terminal in this sequence can be rewritten using rule (4):

```
< [mary], [was], [kissed], pp(_)/by(_,john) >
```

Rewriting this sequence using rule (11) gives:

```
< [mary], [was], [kissed], [by], np/john >
```

which leads to the final result:

```
< [mary], [was], [kissed], [by], [john] >
```

This strategy can be characterized as a left-to-right recursive descent (top-down) strategy. At some points during the rewriting we could have applied other rules too. In fact, another solution can be obtained by using rule (5) in the second step of the derivation. This is the case, because the np to be rewritten is not yet labelled with semantic information. But this implies that we could have applied rule (7) as well, giving:

```
< [the], n/Subj, vp(Subj)/kissed(john,mary) >
```

Clearly this introduces a wasteful sub-computation. The intuition is clear: derivations should only be attempted for non-terminals for which definite information concerning their logical forms is available. Another – related – problem can be illustrated by rewriting the `n/Subj` non-terminal. This might be done using rule (8), giving rise to a sequence that starts with an identical non-terminal. This might lead to non-termination, just like simple top-down parsers lead to non-termination for left-recursive grammars.

Such observations motivate the introduction of more complex generation strategies. There are two ways to go. We might either change the grammar (automatically) so that such situations are carefully avoided, or, alternatively, we might devise more clever interpreters for such grammars. The difference between these two approaches is only a superficial one. Compilation techniques are for example discussed in [19, 6]. We restrict the discussion to different ways of interpreting the grammar, because it simplifies the presentation.

Classifying generation strategies. The different approaches towards the syntactic generation problem can be classified according to a number of dimensions. It is helpful to distinguish between

- Definition of the search space
  - Left-right vs. Bidirectional processing
  - Top-down vs. Bottom-up processing.
- Traversal of the search space

A generator proceeds from left to right if the elements of the right-hand-side of a rule are processed in a left-to-right manner. This order of processing is very common for parsing, but turns out to be unsuitable for generation. The reason is that the important information that guides the generation process, namely the logical forms, is usually percolated in a different manner. For this reason semantic-head-driven generation approaches have become popular. Such approaches aim at an order of processing in which an element of the rhs of a rule is only processed once its corresponding logical form has been determined.

As in parsing theory, generation techniques can be classified according to the way they construct 'parse-trees'. Bottom-up and top-down traversals have been proposed, as well as mixed strategies. As in parsing, bottom-up approaches solve some non-termination problems that are encountered in certain top-down procedures.

The above mentioned two dimensions characterize the way in which parse trees are constructed. A particular choice of these parameters defines a non-deterministic generation scheme, giving rise to a search space that is to be investigated by an actual generation algorithm. Hence generation algorithms can be further classified with respect to the search strategy they employ. For example, a generation algorithm might propose a depth-first backtrack strategy. Potentially more efficient algorithms might use a 'chart' to represent successfull branches of the search space, optionally combined with a breadth-first search.

1.3 Semantic-head-driven generation

In this section I present in short a specific generation algorithm based on [15]. Variants of this algorithm have been used in several projects. [2]

According to the dimensions discussed in the previous section semantic-head-driven generation proceeds in a bidirectional way (semantic-heads first), using a mixture of bottom-up and top-down processing. Different search methods for semantic-head-driven generation are possible.

Semantic-head-driven generation aims at a traversal of the parse-trees that reflects the information flow of the logical forms that are built up. Semantic-head-driven generation turns out to be especially useful for lexicalist linguistic analyses such as those proposed within HPSG and unification-based versions of Categorial Grammar.

[2] [1, 3, 5, 10, 13]

An important motivation for semantic-head-driven generation are the existence of rules in which the semantics associated with the left-hand-side is identical to the semantics of one of the elements of the right-hand-side. Such an element is called the *semantic-head* of the rule. In our example grammar rule (1) the 'vp' non-terminal classifies as the semantic head. Rules that have a semantic-head are called 'chain-rules'. All other rules (including lexical entries) are called non-chain-rules.

In order to generate a parse tree for a non-terminal g the algorithm selects a non-chain-rule. The semantics of the lhs of this non-chain-rule is unified with the semantics of g. This node functions as a 'pivot'. The parse-tree is expanded in a top-down fashion below the pivot, and in a bottom-up fashion above the pivot.

Example. To be somewhat more concrete, consider a simple Prolog implementation of the algorithm. Here we assume that the categories of our previous example are now written as `x(Syn,Sem,P0,P)` to make the contribution to the output string explicit for each non-terminal. This is useful because our traversal will not be strictly from left to right anymore.

```
generate(Sem,String) :- goal(x(s,Sem,String,[])).

semantics(x(_,Sem,_,_),Sem).

goal(Goal) :-
    semantics(Goal,Sem), semantics(Pivot,Sem),
    non_chain_rule(Pivot,Ds), generate_ds(Ds),
    connect(Pivot,Goal).

connect(X,X).
connect(Head,Goal):-
    chain_rule(Head,Lhs,Ds), generate_ds(Ds),
    connect(Lhs,Goal).

generate_ds([]).
generate_ds([H|T]) :-
    goal(H), generate_ds(T).
```

Consider again the example `s/kissed(john,mary)`. This relates to the goal (for brevity I abbreviate the logical forms) `?- goal(x(s,k(j,m),P0,[]))`. In order to satisfy the algorithm selects a non-chain-rule of which the semantics of the lhs matches with the semantics of the goal. We have two possible candidates: rule (3) and rule (4). Rule (3) will eventually lead to the answer 'john kissed mary'. Consider what happens if rule (4) is selected. We then need to generate the remaining element of the rhs of rule (4) (the `pp`), after which the `vp` should be connected up to the goal category.

The generation of the goal `goal(pp,by(_,john),Q0,Q)` proceeds as follows. Again a non-chain-rule is selected. Rule (11) is the only possibility. This leads to the situation in which we need to generate the remaining element of rule (11) (the `np`), after which the lhs of rule (11) needs to be connected to the `pp` goal. It is not difficult to check that this eventually leads to the following instantiation of the goal: `goal(pp,by(_,john),[by,john|Q],Q)`.

At this point the `vp` should be connected to the goal `s`:

```
connect(x(vp,k(j,m),[was,kissed,by,john|Q],Q),
        x( s,k(j,m),P0,[])).
```

In order to connect a category c to a goal g a chain-rule is selected of which the head matches c. Other elements of the rhs of this rule are generated recursively, and the lhs of this rule is then connected to g. In this case, the `vp` matches the head of rule (1) (a chain-rule). The subject `np` should be generated, and then the lhs of rule (1) (`s`) is connected to the goal category `s`. Again, it is straightforward to check that this can be accomplished easily, instantiating `P0` as the generated sentence `[mary,was,kissed,by,john]`.

In this example, only the first invocation of the `goal` predicate led to a choicepoint (related to the choice for active or passive voice). No other branching of the search space was neccessary. Also note that in each of the recursive calls of the `goal` predicate the logical form indeed is instantiated.

Properties. The algorithm defines a simple bottom-up head-driven generation procedure. The search is guided by the input (logical forms) because of the definition of goal `predicate`. This definition implements the fact that heads always share the logical form with their mother (by definition). Therefore, for any given top-goal it is possible to share the logical form with the 'pivot' before the parse tree connecting these two categories is constructed. The resulting system implements, therefore, an attractive compromise between bottom-up and top-down approaches; the order of processing in the presence of headed rules is bottom-up, but there is an important information flow in top-down direction.

Note that the order of processing of the algorithm is not left-to-right, but *bi-directionally* because the algorithm always starts from the head of a rule. The logical form of this head is supposedly known by top-down guidance. This constitutes the top-down information of the algorithm. Apart from the top-down logical form information, the algorithm is directed by the information of the lexicon because the order of processing is bottom-up if the rules are headed. Head driven bottom-up generators are thus geared towards the semantic content of the input on the one hand and lexical information on the other hand. Of course this is especially useful for grammars that are written in the spirit of *lexicalistic* linguistic theories such as CUG, UCG and HPSG.

Apart from considerations of efficiency the major reason for constructing *bottom-up* generators has been the left-recursion problem shortly explained above. If the base case of the recursion resides in the lexicon the bottom-up approach does not face these problems.

Limitations of semantic-head-driven generation. Certain linguistic analyses do not adhere to the implicit assumptions of semantic-head-driven generation. For example, one of the implicit assumptions is that once a head has been constructed, its logical form should be known. In [20] it is shown that a straightforward analysis of verb-second in terms of gaps and gap-threading neccesitates an extension of the simple semantic-head-driven generator.

Other linguistic phenomena such as partial vp-topicalisation may lead to further adaptations of the simple generation regime illustrated above.

1.4 Related issues

Clearly the discussion above neccessarily neglects a number of interesting problems (and partial answers) related with syntactic generation. Some of these are shortly described in this section.

Logical equivalence. An important theoretical and practical problem for natural language generation is the problem of logical form equivalence. For a discussion of this problem we refer to [14].

Incremental generation. Rather than generating on the basis of a single complete logical form, some researchers have investigated the possibility of generating *incrementally*. In such a model small pieces of semantic information are provided to the tactical generator one at the time. Such a model might better explain certain psycholinguistic observations concerning human language production (cf. for example [16]).

Producing a non-ambiguous utterance. The generation procedures sketched above all come up with a possible utterance for a given meaning representation. However, given that natural language is very ambiguous chances are that this proposed utterance itself is ambiguous, and therefore might lead to undesired ‘side-effects’. Some preliminary techniques to prevent the production of ambiguous utterances are discussed in [12].

References

[1] Hiyan Alshawi and Stephen G. Pulman. Ellipsis, comparatives, and generation. In Hiyan Alshawi, editor, *The Core Language Engine*, pages 251–275. ACL-MIT press, 1992. chapter 13.

[2] Douglas E. Appelt. Bidirectional grammars and the design of natural language generation systems. In *Theoretical Issues in Natural Language Processing 3*, pages 206–212, New Mexico State University, 1987.

[3] Hans Ulrich Block. Compiling trace & unification grammar for parsing and generation. In *Proceedings of the ACL workshop Reversible Grammar in Natural Language Processing*, Berkeley, 1991.

[4] Stephan Busemann. *Generierung natuerlicher Sprache mit Generalisierten Phrasenstruktur-Grammatiken.* PhD thesis, University of the Saar, 1990. Also available as TU Berlin, Dept. of Computer Science, KIT report 87.

[5] Jonathan Calder, Mike Reape, and Henk Zeevat. An algorithm for generation in unification categorial grammar. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 233–240, Manchester, 1989.

[6] Marc Dymetman. *Transformations de Grammaires Logiques et Réversibilité en Traduction Automatique.* PhD thesis, Institut IMAG, Université Joseph Fourier (Grenoble 1) & CNRS, 1992.

[7] Marc Dymetman and Pierre Isabelle. Reversible logic grammars for machine translation. In *Proceedings of the Second International Conference on Theoretical and Methodological issues in Machine Translation of Natural Languages*, Pittsburg, 1988.

[8] Clare Gardent and Agnes Plainfossé. Generating from a deep structure. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, Helsinki, 1990.

[9] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar.* Blackwell, 1985.

[10] Dale Douglas Gerdemann. *Parsing and Generation of Unification Grammars.* PhD thesis, University of Illinois at Urbana-Champaign, 1991. Cognitive Science technical report CS-91-06 (Language Series).

[11] Paul S. Jacobs. Achieving bidirectionality. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

[12] Günter Neumann and Gertjan van Noord. Reversible grammars for self-monitoring and generation of paraphrases. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing.* Kluwer, 1993.

[13] Graham Russell, Susan Warwick, and John Carroll. Asymmetry in parsing and generating with unification grammars: Case studies from ELU. In *28th Annual Meeting of the Association for Computational Linguistics*, University of Pittsburg, 1990.

[14] Stuart M. Shieber. The problem of logical form equivalence. *Computational Linguistics*, 19(1), 1993.

[15] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. Semantic-head-driven generation. *Computational Linguistics*, 16(1), 1990.

[16] Koenraad De Smedt and Gerard Kempen. Incremental sentence production, self-correction and coordination. In Gerard Kempen, editor, *Natural Language Generation*, pages 365–376. Martinus Nijhoff, Dordrecht, 1987.

[17] Tomek Strzalkowski. Automated inversion of a unification parser into a unification generator. Technical report, Courant Institute of Mathematical Sciences, New York University, 1989. technical report 465.

[18] Tomek Strzalkowski, editor. *Reversible Grammar in Natural Language Processing*. Kluwer, 1993.

[19] Tomek Strzalkowski and Ping Peng. Automated inversion of logic grammars for generation. In *28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, 1990.

[20] Gertjan van Noord. *Reversibility in Natural Language Processing*. PhD thesis, University of Utrecht, 1993.