GENERATION FROM UNDER- AND OVERSPECIFIED STRUCTURES*

DIETER KOHL
Universität Stuttgart
Institut für maschinelle Sprachverarbeitung
– Computerlinguistik –
Azenbergstraße 12
D-7000 Stuttgart 1
Germany
EMAIL: dieter@adler.philosophie.uni-stuttgart.de

Abstract

This paper describes informally an algorithm for the generation from under- and overspecified feature structures. The generator requires a grammar, a goal category and a feature structure as input, and derives all strings whose corresponding feature structure is not in contradiction to the input structure.

1 Introduction

In this paper I will present an algorithm for generation from under- and overspecified feature structures in the LFG framework[1]. The algorithm makes use of the concept of *generation as structure-driven derivation* as it is described in [14, 15, 16]. Most of the time the algorithm works top-down breadth-first, similar to the generator described in [7] and [6]. Only for the creation of the final structure the algorithm works bottom-up.

2 Motivation

The algorithm given in [14] allows to generate from a fully specified feature structure, e.g. the input structure is equal to a structure that would be derived during parsing. For applications other than testing a grammar for overgeneration the equality-condition is too restrictive.

The algorithm given in [15] and [16] then allows to generate from an underspecified structure, if there is a fully specified (semantic) predicate-argument-structure which is not allowed to be extended during generation, e.g. the predicate-argument-structure must be complete and coherent with respect to the target grammar. One of the disadvantages of this algorithm is, that it must be marked for the generator, which substructure is not allowed to be changed during generation. Further, in certain applications, the condition that there is a partial feature structure which is complete and coherent with respect to the target grammar might be also too restrictive.

The generator described in this paper had been developed for projects which are involved in machine translation. While one of the projects makes use only of syntactic information encoded in a feature structure the other project uses semantic information as well. In both cases the input feature structure for the generator is at least underspecified with respect to the target grammar, not only for atomic attribute value pairs but also for complex pairs. This means the generator has to introduce information into the given feature structure to get a structure which is valid with respect to the target grammmar.

In both projects a similar architecture is used:[2]

1. parse a sentence and return the feature structure F_p
2. extract the information for the translation from F_p and build F_g
3. generate from F_g a sentence

In such an architecture the creation of F_g is usually independent of the target grammar, in the sense that the creation is not automatically controlled by the target grammar.

In machine translation the grammars used for parsing and for generation are basically specific for the two single languages one wants to translate between. It is usually desirable to specify F_g only in as rudimentary and as general manner as possible. This means the details of how to generate a valid surface string of the target language are only known in the target grammar, rather than spelled out in the translation relation. In other words, a single grammar G describes only the relation of a surface string of a language L and a feature structure valid for the grammar G of L. Further, a valid feature structure for G will represent only information necessary for L, but not necessarily information necessary for the language to translate into. For example, a grammar for German will describe a feature structure which has information for the tenses *past*, *present*, and *future*, but no information about *progressive* as it is required for English. Therefore, in the translation German to English the generator has to generate from a feature structure which might be *underspecified* with respect to tense information, while in the translation English to German the generator has to generate from a feature structure which might be *overspecified* with respect to tense information.

In general, in describing the translation relation between two languages one has to face the problems of interfaces:

- Information is missing and must be derived from the target grammar, e.g. the input structure is *underspecified*.

*The work reported here is part of the Sonderforschungsbereich 340 *Sprachtheoretische Grundlagen der Computerlinguistik*

[1] For details of the LFG formalism see [1].

[2] For the reasons of this architecture see for example [4]. There are also other MT projects like GRADE (see [9], [10] and [8]) which make use of a similar architecture.

- There is more information than defined by the target grammar, e.g. there is no string of the target language for which the grammar describes a feature structure which contains all attribute-value pairs given in the input structure FS_g. The input structure is *overspecified* and the overspecification could be ignored during generation.
- There is information which is inconsistent with the target grammar, e.g. the input structure is *illformed* with respect to the target grammar. This requires some error treatment.

An algorithm for generation then has to provide mechanisms which allow generation from underspecified structures as well as from overspecified ones. This will allow to deal with certain types of *translation mismatches* as they are described for example in [2]. Further, the treatment of illformed structures should be such, that the invalid elements of the input structure could be made visible for debugging purposes, instead of just failing to generate anything. As it turned out, even for medium sized grammars it can become quite difficult for a linguist to debug the grammar if there is only a debugger available which had been developed for the general purpose programming language the system is implemented in, e.g. prolog.

3 Terminology

The algorithm has been developed for grammars written in the LFG-formalism. This means, it works on a context-free grammar G with annotated feature descriptions. Given a feature structure FS_{in} as input the algorithm has to generate all those surface strings, for which G associates a feature structure FS_g, with FS_{in} compatible to FS_g.

What *compatible* means depends on the kind of application the generator is used in:

- If the application is to test a grammar for overgeneration, FS_{in} must be equal to FS_g, e.g. no information is introduced into or deleted from FS_{in} during generation, and FS_{in} unifies in terms of feature unification with FS_g.
- If the application is to test whether a structure of a certain attribute might be sufficient for generation, i.e. whether the semantic structure does not overgenerate, FS_{in} must be subsumed by FS_g, e.g. all information of FS_{in} must be required for generation, and it is only allowed to introduce information into FS_{in}.
- If the application is machine translation, FS_{in} and FS_g must unify, e.g. FS_{in} might contain more information and also less information than FS_g.

Depending on the application the algorithm is parametrized as to whether it allows the introduction of information into FS_{in} and whether it allows FS_{in} to be overspecified.

For those not familiar with LFG I will give a short overview of the elements of the feature descriptions as I will use them afterwards. In general a feature description consists of a conjunction of equations or a disjunction of feature descriptions. In this paper I will only consider feature descriptions without disjunctions. The equations are distinguished into

- defining equations indicated by the operator $=$
- inequations indicated by the operator $\neq$
- constraining equations indicated by the operator $=_c$

An equation consists of a reference to a structure, the operator, and as second argument of the operation one of

- an atomic value like `mas`
- a *semantic form*, indicated by double quotes, with an atomic name and an optional argument list, i.e. "man", "give ⟨SUBJ,OBJ⟩"
- a reference to a structure

A reference to a structure is either a meta-variable or a path applied to a meta-variable. Examples are

- the meta-variable $\uparrow$, which stands for the structure associated with the mother node, e.g. the category given on the left hand side of a rule.
- the meta-variable $\downarrow$, which stands for the structure associated with a daughter node of a rule, e.g. the node on the right hand side of a rule where the feature description is an annotation of.
- ($\uparrow$ GENDER), which refers to a structure under the attribute GENDER in the feature structure associated with the mother node.

Equations, which have references on both sides of a equation are called *reentrancy equations*.

Semantic forms describe unique values, e.g. while two atomic values unify if they are described by the same form, two semantic forms will not. The arguments of a semantic form of an attribute A are paths which are members of the *governable functions of* A. This set will be named as gf(A). To allow semantic forms as possible values for any attribute is a generalization of the use of semantic forms as they are given in [1] where semantic forms are only values of the attribute PRED. Semantic forms contain all information necessary to test the conditions of completeness and coherence.

3.1 Coherence and Completeness

Using the generalization the conditions of completeness and coherence as given in [3, pp. 211/212] are reformulated as

- A feature structure S is locally complete iff for each attribute A in S where gf(A) is non-empty the governable functions defined by the value of A exist in S with a value for the attribute A, and if all values required are defined. A structure is complete if all of its substructures are locally complete.
- A feature structure S is locally coherent, iff for each attribute G of S which is member of gf(A) G is governed by the value of A, e.g. the argument list of the value of A contains G, and if all attributes of S are given by the grammar. A structure is coherent if all of its substructures are locally coherent.

The structure FS_g derived in the generation process must at least fullfill these conditions of completeness and coherence, e.g. any violation of one of these conditions is treated as an error. Since the input structure FS_{in} should be part of the derived structure, the conditions for attribute-value pairs of the input structure are modified to be able to use the input structure to control the generation process and to be able to allow overspecification.[3]

- If an attribute A of FS_{in} is licensed by a defining equation or inequation in the rules of the grammar which are not explicitly excluded by FS_{in} it should be checked that A is actually consumed during generation. This condition extends the condition of completeness.
- If an attribute A of FS_{in} does not occur in any equation of the grammar, the input structure is overspecified. It depends on the application, whether this type of overspecification is allowed, e.g. whether it should be considered as a violation of the coherence condition or should be ignored.
- If an attribute A of FS_{in} is not licensed by a defining equation or an inequation in the rules of the grammar which are not explicitly excluded by FS_{in} the input structure is overspecified. It depends on the application whether this type of overspecification is allowed. In case overspecification is allowed, A and its value are ignored, otherwise it is treated as a violation of the coherence condition.

As indicated by the last extension to the coherence and completeness conditions, it depends on the application what kind of input structure is considered to be a valid one for the target grammar. In case a grammar should be tested for overgeneration a valid input structure is not allowed to be extended during generation and is not allowed to be overspecified.

In the case of machine translation the input structure can be considered as a valid one, even it is underspecified. Depending on the language pair it might be also appropriate to consider an overspecified input structure as valid.

4 The Algorithm

The algorithm works on a grammar description and an input feature structure. The grammar description consists of context free rules with annotated feature descriptions.

For simplicity it is assumed that the annotated feature descriptions do not contain disjunctions. A disjunction in a feature description can always be transformed into a disjunction of nodes on the c-structure level. Furthermore, a single rule is a concatenation of terminal and non-terminal nodes, and for each category C of a grammar the rules for C are treated as one disjunction.

[3]This means, it is not sufficient to require, that the input structure has to unify with a structure derived from the grammar to get a generation, since this would allow to produce sentences which do not contain all of the semantics given in the input structure as well as to produce sentences with any kind of possible modifiers the grammar could derive, that is infinite many.

The algorithm starts with a current category C_c, initialized with the goal category, and a current feature structure FS_c, initialized with the input feature structure FS_{in}.

The algorithm proceeds as follows:

- *Match* the current feature structure FS_c with the current category C_c by matching FS_c with the feature descriptions FD_i of the nodes Ni on the right hand side of the rule for C_c, where FS_c is bound to the mata variable $\uparrow$ which denotates the structure associated with the mother node C_c on the left hand side. The matching works top-down breadth-first. During the match FS_c will not be modified.
- *Extend* FS_c by the application of a feature description FD.

4.1 Matching

The matching of the current feature structure FS_c with the current category C_c will always terminate. During the matching a structure which is used as a chart and an agenda is built which keeps track of

- which structures are already matched with which categories.
- whether there occurs a ***trivial recursion***, e.g. given a structure and a category there is a recursion on the c-structure level which uses the same structure.
- the use of which nodes can be constrained by the input structure, and what is the result, e.g. is the usage of the node excluded or licened by the input structure.
- which nodes are purely controlled on the c-structure level, e.g. there is no equation for a node which denotates the structure of the mother node. Such nodes have to produce only finite many substrings.

For each category C all its rules are considered in parallel, which avoids any dependency about the ordering of the single rules for C.

For each node N on the right hand side of C_c the input feature structure is matched with its feature description FD. This match results in at least one of the following descriptions:

Exclusion: FS_c is not compatible with FD. Therefore the node N will be excluded. Other results of the matching are of no relevance. The exclusion of N excludes those nodes which are part of the same rule as N.

Activation: FD defines a path-value-pair which is already part of FS_c, or FD defines a reentrency which already exists in FS_c.

Examination: In FD occurs a reentrance equation where only one of the paths exists in FS_c. The result *examination* contains the category C_N named by the node N and the associated substructure FS_s.

The following cases are distinguished:

trivial equation: N is a non-terminal node. The catgories C_c and C_N are associated with the same (sub)structure. Beside $\uparrow = \downarrow$ equations of the form $(\uparrow \mathrm{X}) = (\downarrow \mathrm{X})$ are also considered as trivial equations.

$(\uparrow \mathrm{X}) = \downarrow$: N is a non-terminal node. The *category* C_N will be matched with the structure denotated by $(\uparrow \mathrm{X})$.

$(\uparrow \mathrm{X}) = (\downarrow \mathrm{Y})$: N is a non-terminal node. The *category* C_N will be matched for $(\downarrow \mathrm{Y})$ with the structure denotated by $(\uparrow \mathrm{X})$. This case covers the treatment of multiple rooted structures as they might occur in grammars written in an HPSG style[4].

$(\uparrow \mathrm{X}) = (\uparrow \mathrm{Y})$: C_c will be matched for $(\uparrow \mathrm{Y})$ with the structure denotated by $(\uparrow \mathrm{X})$.

Uncontrolled: *FD* does not contain any equation which can be applied on FS_c. In this case FS_c does not controll the occurence of the substring associated with the node N, and it depends on the partial *c*-structure alone given by the category C_N whether there are finite many substrings described.

Suspension: *FD* contains equations which allow controll of generation by FS_c, but FS_c does not contain enough information to make a decision about exclusion, activation or examination. Therefore, the matching of N with FS_c has to be decided later. In case the application forbids introduction of information into FS_c during generation the conditions of suspension will lead to immediate exclusion.

Only the results *activation* and *examination* may occure in parallel. The result *examination* causes a further examination of the category C_N with the selected (sub)-structure, if they have not already been examined and are not already under examination. Thus the matching of a category with a (sub)-structure is performed only once during the matching of the input feature structure with the goal category. This guarantuees the termination of the matching and is efficient.

Since the matching works top-down breadth-first it is possible to detect inconsistencies between the input feature structure and parts of the rules fairly early.

From the complete match it is possible to determine the set of these attribute-value pairs, which are part of the original input structure and which *could* be used either by a defining equation or an inequation. These attribute-value pairs are marked that they have to be used which is an equivalent of adding temporarely constraining equations to the grammar. which guarantee that a maximum of information from the input structure is used for generation. It should be noted, that this step is only necessary, if overspecification of the input structure is allowed. Otherwise all attribute value pairs of the input structure could be marked at startup that they have to be used during generation.

The matching produces a set of possible solutions. This makes it possible to distinguish a failure caused by an illegal input structure from the generate-and-test behaviour of the backtracking mechanism. Since there is enough information of the current goal in the generation process, it is possible to produce an error message which describes

- the *c*-structure build so far
- the node and its annotated feature description which is inconsistent with the input structure
- the part of the input structure which caused the failure

Such an error message would be in terms of the grammar rather than in terms of the implemention language of the algorithm. An error message could be

> *I couldn't generate an NP for the structure* $\begin{bmatrix}\text{PRED man}\\ \text{SPEC idef}\end{bmatrix}$ *because* SPEC = idef *is illegal for the grammar.*

Since it is distinguished which parts of the structure are introduced during generation it is possible to show only those failures which are caused by the original input structure. This would also allow one to ignore illegal parts of the input structure completely and to even generate from illformed structures. In contrast to the case of overspecification this would require repairing either the input structure or extending the target grammar.

4.2 Extension

The extension of FS_c by a feature description *FD* means, that all information from *FD* is incorporated into FS_c. Since only non-disjunctive feature descriptions are considered it is not necessary to describe the treatment of disjunctive information. The only source of alternatives are the rules. These alternatives are treated by backtracking. The selection of alternatives starts with those disjuncts, which do not lead to recursion. This guarantees that recursion is applied only in those cases, where it could be part of the *c*-structure to generate.

The extension has several aspects. First, it is made explicit in the feature structure which attribute-value pairs are defined by the grammar, and how often a definition has occured during the generation. The latter information is used to stop the generation from infinite loops by giving a maximum amount of repeated definitions of the same piece of information. Reasonable limits are values between 10 and 20. It should be noted that the *semantic forms* of LFG reduce this limit to 1 for attributes which take a semantic form as value[5].

Second, a partial representation of the *c*-structure is built in parallel to the feature structure, which allows at the end of the generation process to extract the surface string by a traversal of the complete *c*-structure.

Third, it can be determined which attribute-value pairs have been introduced into the original structure. Only these attribute-value pairs are relevant to reexamine suspended nodes.

[4]For a description of HPSG see [11].

[5]For LFG grammars this aspect of semantic forms is the main reason that the generation will terminate without the superficial limitation of repeated definitions.

4.3 The main loop

1. For each node N_j of the right hand side of the rule of the current category C_c match the annotated feature description FD_j with the current feature structure FS_c. The matching terminates always, and during the matching no new information is introduced into FS_c. The match determines, whether the node N_j might be *excluded*, *activated*, *suspended*, and whether the category N should be *examined* for some part of FS_c.
2. If there are no nodes left which can be activated, nodes which are still suspended are excluded and the final coherence and completeness tests are performed on the input structure FS_{in}. In case of success the surface string can be extracted from the c-structure which is built in parallel to the derivation of the input feature structure. In case of failure, other solutions are tried by backtracking.
3. Select only these nodes which can be activated which will not lead to a recursion. *Extend* the partial feature structures associated with these nodes by applying the annotated feature descriptions.
4. Compare those nodes again which have been suspended as in step 1.
5. Repeat the steps 3 and 4 until there are no nodes left which can be activated and which do not lead to a recursion.
6. Nodes which could be activated but lead to recursion are activated only in case there is no indication that the recursion could be applied infinite many times[6].
7. Continue with step 2.

5 Example

In order to illustrate how the algorithm works, I will only give a very simple and somewhat superficial example. For more detailed examples especially on the treatment of recursion see [5].[7]

The example makes use of the grammar in figure 1 to generate a German sentence with a simple NP and an intransitive verb. The grammar is written in a usual LFG notation. The input feature structure for generation is given in figure 2. For the example it is assumed that the feature stucture contains the semantic representation of the analysis of the English sentence *the man is running* which should be translated into German. The goal category for generation is **S**.

The generation starts with the matching of **S** with FS_0. The **NP** node of the right hand side of the **S** rule is suspended, since there is no attribute SUBJ in the input structure. The trivial equation of the **VP** node immediately leads to the matching of FS_0 with the category **VP**. The trivial equation on the **V** node leads in turn to the matching of the category **V** with FS_0. The existence of (SEM REL) = *run* in FS_0 would allow to activate both verbs of the example lexicon, but the equation (↑ SEM TIME END) = past excludes the entry for *rannte*.

[6] In this paper infinite loops are only assumed in case the limit of repeated definitions is reached. A more detailed treatment of the detection of infinite loops is given in [5]

[7] There would be not enough space to show a more complicated example in this paper.

$$\mathrm{S} \rightarrow \begin{array}{cc} \mathrm{NP} & \mathrm{VP} \\ (\uparrow \mathrm{SUBJ}) = \downarrow & \uparrow = \downarrow \end{array}$$

$$\mathrm{NP} \rightarrow \begin{array}{cc} \mathrm{D} & \mathrm{N} \\ \uparrow = \downarrow & \uparrow = \downarrow \end{array}$$

$$\mathrm{NP} \rightarrow \begin{array}{c} \mathrm{N} \\ \uparrow = \downarrow \end{array}$$

$$\mathrm{VP} \rightarrow \begin{array}{c} \mathrm{V} \\ \uparrow = \downarrow \end{array}$$

mann:	N,	(↑ PRED) = "mann"
		(↑ NUM) = sg
		(↑ GENDER) = mas
		(↑ CASE) ≠ gen
		(↑ SEM REL) = "man"
		(↑ SEM NUM) = sg
der:	D,	(↑ SPEC) = def
		(↑ GENDER) = mas
		(↑ CASE) = nom
		(↑ NUM) = sg
		(↑ SEM SPEC) = def
rennt:	V,	(↑ PRED) = "rennen ⟨SUBJ⟩"
		(↑ TENSE) = present
		(↑ SUBJ CASE) = nom
		(↑ SUBJ NUM) = sg
		(↑ SEM REL) = "run"
		(↑ SEM TIME START) = now
		(↑ SEM ARG1) = (↑ SUBJ SEM)
rannte:	V,	(↑ PRED) = "rennen ⟨SUBJ⟩"
		(↑ TENSE) = past
		(↑ SUBJ CASE) = nom
		(↑ SUBJ NUM) = sg
		(↑ SEM REL) = "run"
		(↑ SEM TIME START) = past
		(↑ SEM TIME END) = past
		(↑ SEM ARG1) = (↑ SUBJ SEM)

Figure 1: Example grammar

The resulting partial c-structure of the match is

```
S—NP ...suspended...
  VP—V—"rennt"
```

The following attribute value pairs of FS_0 must be used during generation:

(SEM REL)
(SEM ARG1)
(SEM TIME START)

Since the solution set of the match does *not* require to use (SEM TIME END) this information can be ignored for the further generation, although it had been used to exclude an entry. This shows a case of overspecification, where an attribute is in the set of possible attributes of a grammar but is not always determined by the grammar.

The extension of FS_0 then leads to the structure in figure 3. It should be noted that the algorithm automatically selected the semantic head, although

$$\boxed{0}\begin{bmatrix} \text{SEM} & \boxed{1}\begin{bmatrix}\text{REL} & \text{"run"}\\ \text{ARG1} & \boxed{2}\begin{bmatrix}\text{REL} & \text{"man"}\\ \text{SPEC} & \text{def}\\ \text{NUM} & \text{sg}\end{bmatrix}\\ \text{TIME} & \boxed{3}\begin{bmatrix}\text{START} & \text{now}\\ \text{END} & \text{future}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 2: Input structure for generation

the head is embedded in a substructure. This means the algorithm is implicit head-driven without any assumptions which part of an input structure the head should be. As it is shown in [5], this allows to generate in cases of head-switching, where syntactic and semantic head differ.

$$\boxed{0}\begin{bmatrix} \text{SEM} & \boxed{1}\begin{bmatrix}\text{REL} & \text{"run"}\\ \text{ARG1} & \boxed{2}\begin{bmatrix}\text{REL} & \text{"man"}\\ \text{SPEC} & \text{def}\\ \text{NUM} & \text{sg}\end{bmatrix}\\ \text{TIME} & \boxed{3}\begin{bmatrix}\text{START} & \text{now}\\ \text{END} & \text{future}\end{bmatrix}\end{bmatrix}\\ \text{PRED} & \text{"rennen ⟨SUBJ⟩"}\\ \text{TENSE} & \text{present}\\ \text{SUBJ} & \boxed{4}\begin{bmatrix}\text{CASE} & \text{nom}\\ \text{NUM} & \text{sg}\\ \text{SEM} & \boxed{2}\end{bmatrix}\end{bmatrix}$$

Figure 3: First extension of the input structure

The introduction of SUBJ leads to the matching of the suspended **NP** node with FS_0. The equation $(\uparrow \text{SUBJ}) = \downarrow$ leads to the matching of the category **NP** with FS_4.

For the **NP** rule there are three nodes to be matched with FS_4. Since on all three nodes a trivial equation is annotated, the categories **D** and **N** have to be matched with FS_4. The equations $(\uparrow \text{SEM REL}) = \text{man}$ and $(\uparrow \text{SEM NUM}) = \text{sg}$ activates the noun entry, and requires that (SEM REL) and (SEM NUM) of FS_4 must be used for generation. The equation $(\uparrow \text{SEM SPEC}) = \text{def}$ activates the determiner entry and requires to use (SEM SPEC) of FS_4.

The two alternatives of the **NP** rule allow to consider two possible extension shown in table 1.

Since (SEM SPEC) of FS_4 must be used, the second alternative will be rejected by the final constraint test. Therefore, the only solution is the first alternative. This results in the *c*-structure

```
S—NP—D—"der"
      N—"mann"
  VP—V—"rennt"
```

from which the string *der mann rennt* is generated.

	feature structure	*c*-structure
1.	$\boxed{4}\begin{bmatrix}\text{CASE} & \text{nom}\\ \text{NUM} & \text{sg}\\ \text{SEM} & \boxed{2}\\ \text{PRED} & \text{"mann"}\\ \text{GENDER} & \text{mas}\\ \text{SPEC} & \text{def}\end{bmatrix}$	NP—D—"der" N—"mann"
2.	$\boxed{4}\begin{bmatrix}\text{CASE} & \text{nom}\\ \text{NUM} & \text{sg}\\ \text{SEM} & \boxed{2}\\ \text{PRED} & \text{"mann"}\\ \text{GENDER} & \text{mas}\end{bmatrix}$	NP—N—"mann"

Table 1: Possible extensions of the **NP** rule

6 Comparison with Shiebers approach

The semantic-head-driven algorithm given in [13] also starts with a top-down initalization with a bottom-up generation. In Shieber et al the nodes which contain the semantic head are determined during the compilation of the grammar. This seems to be a bit problematic for grammars which describe head-switching phenomenons, as in *100 litres of wine*, where a possible ananlysis is that *100 litres* syntactically governs *wine*, but semantically is a modifier of *wine*. The algorithm presented here does not require to precompute the nodes which contain the semantic head, but finds the head relevant for the given input structure automatically.

The problem with free variables for the coherence constraint given in Shieber et al does not occur for the algorithm presented in this paper, since it always distinguishes between the structure and the description of the structure, and keeps track of which parts of the structure are already derived during generation. Since the algorithm presented here always has information about which parts are from the original input structure and which ones have been added, it is possible to check the coherence condition at any step of the generation process. In addition, the solution in Shieber et al with binding variables seems somewhat problematic, since it requires to know for sure, that the variable part of the semantics should not be extended.

The augmentation of the generator described in Shieber et al with a chart to avoid recomputation and eliminate redundancies is an integral part of the algorithm presented here.

7 Summary

In this paper an algorithm had been described which can be used to generate from fully specified feature structures as well as from variants of under- or over-specified feature structures in the LFG framework. The algorithm covers the cases given in [14] and [15] as a subset. The treatment of recursion allows even for infinite many possible generations that the solutions can be presented one by one, e.g. the generator will not go into an infinite loop between two solutions.

The generator is implicit head-driven, e.g. it selects the head automatically for a given input structure with respect to the target grammar. As it is shown in

[5] this behaviour of the algorithm allows the efficient treatment of head-switching phenomenons.

It has been shown, that the algorithm provides information which allows in case of failure to produce debugging information in terms of the target grammar, rather than in terms of the programming language the algorithm is implemented in.

The algorithm is implemented in PROLOG in the edinburgh syntax. Currently the implemention of the debugging mechanisms is incomplete.

Although it is not shown in this paper, the technique used for the generator could be easily adopted for parsing, where the input string takes the part of the input feature structure. In this sense the *c*-structure is only considered as an auxiliary structure where the grammar describes basically a relation between a surface string and a feature structure. To adopt the technique for parsing would have the advantages

- to use basically the same machinery for parsing and generation where the machinery is optimized for each task,
- to have the same improved possibilities for debugging, and
- to allow to start the parsing of strings while they are typed in, and not only after the complete string to be parsed is known.

One of the major goals for the future development of the algorithm is to reduce the use of backtracking as much as possible by using disjunctions as part of the feature structure.

The algorithm should be also applicable to other grammar formalisms like PATR-II (see [12]) which make use of a context-free backbone and anotated descriptions. It is also intended to use the algorithm for formalisms like HPSG.

References

[1] Joan Bresnan, editor. *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts, first edition, 1982.

[2] Megumi Kameyama, Ryo Ochitani, and Stanley Peters. Resolving translation mismatches with information flow. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 193–200, Berkley, California, USA, 18–21 June 1989. University of California, Association for Computational Linguistics.

[3] Ronald M. Kaplan and Joan Bresnan. Lexical-functional grammar: a formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, chapter 4, pages 173–281. MIT Press, Cambridge, Massachusetts, 1982.

[4] Ronald M. Kaplan, Klaus Netter, Jürgen Wedekind, and Annie Zaenen. Translation by structural correspondences. In *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, 1989.

[5] Dieter Kohl. Generierung aus unter- und überspezifizierten Merkmalsstrukturen in LFG. Arbeitspapiere des SFB 340 *Sprachtheoretische Grundlagen für die Computerlinguistik* Bericht Nr.9, Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, July 1991.

[6] Dieter Kohl and Stefan Momma. LFG based generation in ACORD. In Gabriel Bes, editor, *The Construction of a Natural Language and Graphic Interface - Results and perspectives from the ACORD project.* Part Generation in ACORD, Chapter 5. Springer, (to appear) 1992.

[7] Stefan Momma and Jochen Dörre. Generation from *f*-structures. In Ewan Klein and Johan van Benthem, editors, *Categories, Polymorphism and Unification.* Cognitive Science Centre, University of Edinburgh and Institute for Language, Logic and Information, University of Amsterdam, Edinburgh and Amsterdam, 1987.

[8] Makoto Nagao. The transfer phase of the mu machine translation system. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 97–103, 1986.

[9] Makoto Nagao, Toyoaki Nishida, and Jun-ichi Tsujii. Dealing with incompleteness of linguistic knowledge in language translation. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 420–427, 1984.

[10] Jun-ichi Nakamura, Jun-ichi Tsujii, and Makoto Nagao. Grammar writing system GRADE of mu-machine translation project and its characteristics. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 338–343, 1984.

[11] Carl J. Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics. Vol. 1 Fundamentals*, volume 13 of *CSLI Lecture Notes.* Univ. Press, Chicago, 1987.

[12] Stuart M. Shieber, Hans Uszkoreit, Fernando C.N. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In B. J. Grosz and M. E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge.* SRI report, 1983. PATR reference.

[13] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. Semantic-head-driven generation. *Computational Linguistics*, 16(1):30–42, March 1990. Refs for bottom-up generation problems.

[14] Jürgen Wedekind. A concept of derivation for LFG. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 486–489, Bonn, West Germany, 25–26 August 1986. Institut für Kommunikationsforschung und Phonetik, University of Bonn.

[15] Jürgen Wedekind. Generation as structure driven derivation. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 732–737, Budapest, Hungary, August 1988.

[16] Jürgen Wedekind. Unifikationsgrammatiken und ihre Logik. Dissertation, Universität Stuttgart, Stuttgart, 1990.