A generation algorithm suitable for f-structure representations

Toni Tuells
IULA , Universitat Pompeu Fabra
La Rambla 30-32, Barcelona
tuells@upf.es

Abstract

This paper shows that previously reported generation algorithms run into problems in dealing with f-structure representations. A generation algorithm that is suitable for this type of representations, the Semantic Kernel Generation (SKG) algorithm is presented. The SKG method has the same processing strategy as the Semantic Head Driven generation (SHDG) algorithm and relies on the assumption that it is possible to compute the Semantic Kernel (SK) and non Semantic Kernel (Non-SK) information for each input structure.
Keywords: tactical generation, reversibility, unification grammars

1 Introduction

Ideally, a declarative grammar should be suitable for both parsing and generation. Unfortunately, things are not that easy. As it is well-known, for many declarative grammars the implicit concept of derivation is that of parsing. As a result, parsing is easier than generation; given a grammar formalism, there are several well-known parsing strategies that one could (more or less easily) adapt. The same does not hold for (tactical) generation, since the task of generating a string out of a semantic representation becomes more complex due to the fact that semantic representations vary according to the theories adopted. Not surprisingly, different generation algorithms are defined for each type of semantic representation. In this paper we take up the problem of generating a string out of an f-structure like representation; instead of designing an interface between this kind of structures and semantic structures useable by already existing generation methods we have decided to implement an algorithm that operates directly on f-structure representations but keeps the main merits of other generation methods as much as possible.[1] We therefore put forward a new generation algorithm, the Semantic Kernel (SK) generation algorithm, that can be seen as a variant of the *semantic head driven* generation algorithm (SHDG) (Shieber *et al.*,1990), (Noord,1993). SKG follows basically the SHDG processing strategy but runs under other assumptions. Since in SKG the *syntactic-head relation* plays also an important role, it can also be seen as a variant of the *syntactic-head driven generation algorithm* (SynHDG) (see (König,1994) and (König,1995)).

2 The Semantic Kernel Generation Algorithm

In order to show how the SKG method works (and why former generation methods are not directly applicable to f-structure representations) we assume the grammar fragment and lexical entries which are given in figures 1, 3.[2] As for the grammar fragment, note that *rules 1a and 1b* introduce modifiers at sentence level, and *rule 3* introduces modifiers at vp level. *Rule 2* combines the subject with the *vp*.

[1] see for example (Busemann,1996) for an alternative proposal on an interface between constraint-based grammars and generation systems.

[2] For the grammar fragment only the relevant semantic information is shown.

Rule 4 deals with the complements of a vp. Note also that the subject is treated as a complement.

$$\begin{bmatrix} \text{CAT:} & \text{v} \\ \text{LEX:} & \text{generate} \\ \text{SUBCAT:} & \langle \text{NP}_{\boxed{1}}, \text{NP}_{\boxed{2}} \rangle \\ \text{SEM:} & \begin{bmatrix} \text{PRED:} & \text{generate} \\ \text{ARG1:} & \boxed{1} \\ \text{ARG2:} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

Figure 1: Lexical entry for *generate*

$$\begin{bmatrix} \text{CAT:} & \text{n} \\ \text{LEX:} & \text{sentence} \\ \text{SEM:} & \begin{bmatrix} \text{REL:} & \text{sentence} \end{bmatrix} \end{bmatrix}$$

Figure 2: Lexical entry for *sentence*

The analysis of the sentence *The little prolog program generated the complex sentence quickly* is given in figure 4.[3] The point is that input for the generator represents the deep predicate argument structure of sentences; modifiers are contained in set-valued feature "mod".[4] Note also that input representations are encoded as (possible typed) Feature Structures.

We first review the direct application of former generation methods to f-structure representations and then we describe more thoroughly the SKG algorithm.

For expository purposes we will use the graphical notation used in (König,1994) for describing the generation algorithms. Taken directly from (König,1994): we will assume that the syntax-semantics-relation for a given grammar is stated by

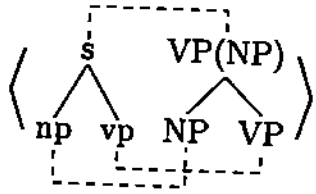
[3]The example is due to Nicolas Nicolov.

[4]We assume that set-valued features are modelled as (Prolog) lists.

$$\begin{bmatrix} \text{MOD:} & \langle \text{quick} \rangle \\ \text{PRED:} & \text{generate} \\ \text{ARG1:} & \begin{bmatrix} \text{DEF:} & + \\ \text{MOD:} & \langle \text{little, prolog} \rangle \\ \text{REL:} & \text{program} \end{bmatrix} \\ \text{ARG2:} & \begin{bmatrix} \text{DEF:} & + \\ \text{MOD:} & \langle \text{complex} \rangle \\ \text{REL:} & \text{sentence} \end{bmatrix} \end{bmatrix}$$

Figure 4: Input semantics

pairs of trees. The left tree states a local syntactic dependency, whereas the right tree defines a local semantic dependency. We also assume that there is a one-to-one mapping from the nonterminal leaf nodes of the syntactic tree on the leaf nodes of the local semantic tree. Note that this is only a graphical notation for the *rule-to-rule hypothesis*, i.e., the fact that in the grammar each syntactic rule is related with a semantic analysis rule. An example is given below:



(1)

The head-corner generator ((Noord,1993), a variant of SHDG) and SynHDG are (graphically) described in figure 5 (taken directly from (König,1994)). The rule *lex* is the *prediction step* of the algorithm, i.e. it restricts the selection of lexical entries to those that can be linked to the local goal (visualized by a dotted line). The rule *hc_complete* is the *bottom-up step* which selects a rule for which xh is the syntactic head and X_h is the semantic head. As a result, it also predicts the head's sis-

(1a) $\begin{bmatrix}\text{cat:} & \text{s}\\ \text{sem:} & \text{SEM}\\ \text{sem:mod:} & [\text{M}\,|\,\text{MODS}]\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{adv}\\ \text{sem:} & \text{M}\end{bmatrix}, \begin{bmatrix}\text{cat:} & \text{s}\\ \text{sem:} & \text{SEM}\\ \text{sem:mod:} & \text{MODS}\end{bmatrix}$

(1b) $\begin{bmatrix}\text{cat:} & \text{s}\\ \text{sem:} & \text{SEM}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{s}\\ \text{sem:} & \text{SEM}\\ \text{sem:mod:} & \text{MODS}\end{bmatrix}, \begin{bmatrix}\text{cat:} & \text{adv}\\ \text{sem:} & \text{M}\end{bmatrix}$

(2) $\begin{bmatrix}\text{cat:} & \text{s}\\ \text{sem:} & \text{SEM}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{np}\\ \text{sem:} & \text{SEM\_SUBJ}\end{bmatrix}, \begin{bmatrix}\text{cat: vp}\\ \text{sem: SEM}\\ \text{subcat:} & [\begin{bmatrix}\text{cat:} & \text{np}\\ \text{sem:} & \text{SEM\_SUBJ}\end{bmatrix}]\end{bmatrix}$

(3) $\begin{bmatrix}\text{cat:} & \text{vp}\\ \text{sem:} & \text{SEM}\\ \text{sem:mod:} & [\text{MOD}\,|\,\text{MODS}]\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{adv}\\ \text{sem:} & \text{MOD}\end{bmatrix}, \begin{bmatrix}\text{cat:} & \text{vp}\\ \text{sem:} & \text{SEM}\\ \text{sem:mod:} & \text{MODS}\end{bmatrix}$

(4) $\begin{bmatrix}\text{cat:} & \text{vp}\\ \text{sem:} & \text{SEM}\\ \text{subcat:} & [\text{SUBJ}\,|\,\text{REST}]\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{vp}\\ \text{sem:} & \text{SEM}\\ \text{subcat:} & [\text{SUBJ}, \begin{bmatrix}\text{cat:} & \text{X}\\ \text{sem:} & \text{SEM}_X\end{bmatrix}\,|\,\text{REST}]\end{bmatrix}, \begin{bmatrix}\text{cat:} & \text{X}\\ \text{sem:} & \text{SEM}_X\end{bmatrix}$

(5) $\begin{bmatrix}\text{cat:} & \text{vp}\\ \text{sem:} & \text{SEM}\\ \text{subcat:} & \text{SUBCAT}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{v}\\ \text{sem:} & \text{SEM}\\ \text{subcat:} & \text{SUBCAT}\end{bmatrix}$

(6) $\begin{bmatrix}\text{cat:} & \text{np}\\ \text{sem:} & \text{SEM}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{det}\end{bmatrix}, \begin{bmatrix}\text{cat:} & \text{n2}\\ \text{sem:} & \text{SEM}\end{bmatrix}$

(7) $\begin{bmatrix}\text{cat:} & \text{n2}\\ \text{sem:} & \text{SEM}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat:} & \text{n}\\ \text{sem:} & \text{SEM}\end{bmatrix}$

(8) $\begin{bmatrix}\text{cat:} & \text{n2}\\ \text{sem:} & \text{SEM}\\ \text{sem:mod:} & [\text{MOD}\,|\,\text{MODS}]\end{bmatrix} \longrightarrow \begin{bmatrix}\text{cat: adj}\\ \text{sem:} & \text{SEM}\end{bmatrix}, \begin{bmatrix}\text{cat:} & \text{n2}\\ \text{sem:} & \text{SEM}\\ \text{sem:mod:} & \text{MODS}\end{bmatrix}$

Figure 3: Grammar Fragment.Only semantic information is shown

ters, which have to be expanded recursively (*top-down prediction*). The difference between SHDG and SynHDG is the link relation for semantic structures: in (Noord,1993) the *semantic-based link relation* is defined as follows:

$$link(X, X_i) \quad if \tag{2}$$

X and X_i are *identical.* If semantics is represented using first order terms, that reduces to check whether X and X_i unify. As for the SynHDG algorithm, the *semantic-based link relation* is defined as follows (König,1994):

$$link(X, X_i) \quad if \tag{3}$$

X_i is a substructure of X. In practical terms, X_i is an element of the bag of semantic keywords that constitute X.

2.1 The direct application of former generation procedures to f-structure representations

We will illustrate the problems of SHDG (more specifically, a variant of it, the head-corner generator described in (Noord,1993)) with f-structures following its application to the (very simple) input semantics given below (which corresponds to the np *the complex sentence*):

$$\begin{bmatrix} \text{cat:} & \text{np} \\ \text{sem:} & \begin{bmatrix} \text{rel:} & \text{sentence} \\ \text{def:} & + \\ \text{mod:} & [\text{complex}] \end{bmatrix} \end{bmatrix} \tag{4}$$

According to the algorithm described in figure 5, and because of the *syntactic-head* and *semantic-head link relation*, the rule *lex* can only be applied to the lexical entry for *sentence* (figure 2). However, since the link relation for semantic structures is defined in terms of *unification* of semantic representations, the application of rule *lex* leads us to a new semantic goal which is *identical* to the input semantics. Now, it comes to apply the rule *hc_complete*; rule *7* is the unique candidate. After applying it, our current goal is the following:

$$\begin{bmatrix} \text{cat:} & \text{n2} \\ \text{sem:} & \begin{bmatrix} \text{rel:} & \text{sentence} \\ \text{def:} & + \\ \text{mod:} & [\text{complex}] \end{bmatrix} \end{bmatrix} \tag{5}$$

At this point, a new *hc_complete* step needs to be performed. Now we have two candidates: rules *6* and *8*. If one selects rule *6*, and after generating recursively the determiner, one ends up having generated only part of the sentence: *the sentence.* On the other hand, rule *8* can be *always* selected; consequently, one could end up having semantic goals that would look like that:

$$\begin{bmatrix} \text{cat:} & \text{n2} \\ \text{sem:} & \begin{bmatrix} \text{rel:} & \text{sentence} \\ \text{def:} & + \\ \text{mod:} & [\text{X,}\ldots\,|\,\text{complex}] \end{bmatrix} \end{bmatrix} \tag{6}$$

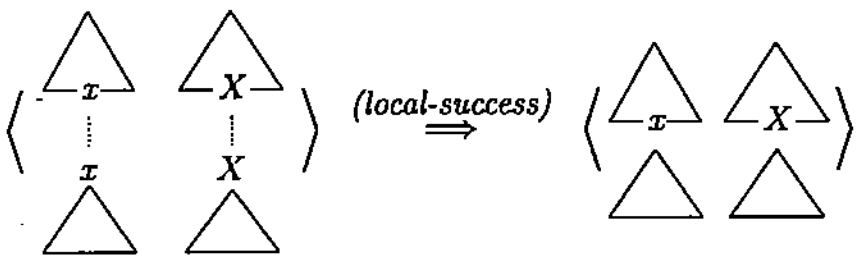
In other words, *the generator would loop and would not terminate.*

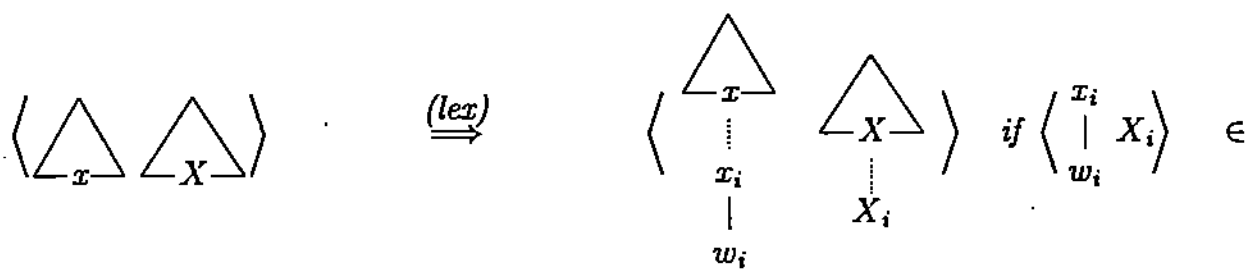
The problems shown above lead us to the following conclusion: the SHDG generator is *neither complete nor coherent.* These issues arise in a similar way for first order terms (see discussion in (Shieber *et al.*,1990)); the problem here is that we do not have a notion of grounded feature structures.

2.2 Semantic Kernel Generator

Although we have found some problems applying SHDG to the grammar in figure 3, it is not very difficult to come up with a *coherent and complete* variant of SHDG . In any case, it seems that the generator needs some knowledge about the semantic structure of a sign. Our first assumption is that the generator is capable of distinguishing between the following different types of semantic information within input structures:

all leaves are labeled with terminals and the tree does not contain any dotted lines *(global-success)*





G *and* $link(\langle x, X\rangle, \langle x_i, X_i\rangle)$

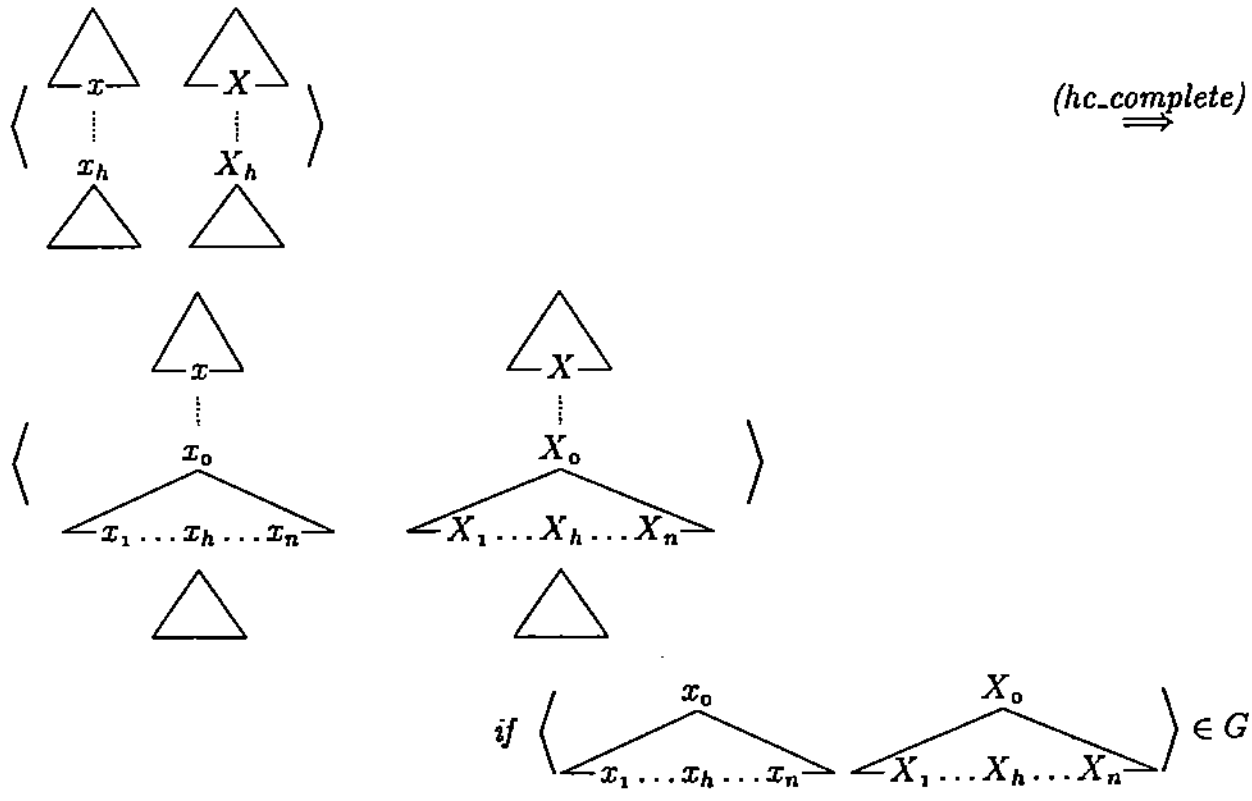


Figure 5: Head-Corner Generator (G grammar description; x_i syntactic category; X_i semantic representation) (Taken from (König,1994))

- **Semantic Kernel (SK) Information:** Semantic structure completely predictible from the lexicon (i.e, there is at least one lexical entry which subsumes this structure).

- **Non Semantic Kernel (Non-SK) Information:** Semantic structure which is not predictible from any lexical entry (typically, lists). In our grammar, modifiers are represented as a list. This list is Non-SK information.

Similarly, the generator is given the following information with respect to the types of rules:

- **SK Rules:** Rules which do not add Non-SK information.

- **Non-SK Rules:** Rules which add Non-SK information. Rules 1,3,8 in our grammar.

The hypothesis behind this classification is that of *structural predictibility*: SK information comes from the lexicon (i.e, SK information can be seen as grounded feature terms), and non SK information is introduced by rules. In other words, the generator knows whether each type of input structure comes from a lexical entry or whether it has been constructed from a (non SK) rule. Thus, the restrictedness of the algorithm results from the fact that it operates under the assumption that one can recursively decompose each input structure into SK and Non-SK information. Although this restriction seems to be rather strong, the generator gives the right answers for the problematic flat f-structure representations containing sentence modifiers.[5] It is worth mentioning that in the SHDG algorithm from (Noord,1993) semantic representations (logical forms) are SK information. In other words, its structure is completely predicted from the lexicon.

A graphical version of the SKG algorithm is given in figure 7 and a simplified prolog-version is given in figure 6. . Further on an example will be given that shows how the algorithm works.

To summarize, this is the information the generator needs to know about the grammar:[6]

- link relation (head relation).

- The SK substructure of a given semantic representation.

- the Non-SK substructure of a given semantic representation.

- The distinction between SK rules and Non-SK rules.

- The syntactic goals for generating SK information.

- The syntactic goals for generating Non-SK information.

- The syntactic goal we obtain after combining SK and Non-SK information.

In order to show how the SKG algorithm works we will follow its application to the input semantics for *the complex sentence* given in example 4. For this input semantics, we have two SK structures:

$$\left[\text{rel: sentence}\right] \quad (7)$$

$$\left[\text{def: } +\right] \quad (8)$$

and one nonSK structure:

$$\left[\text{mods: } \left[\text{complex}\right]\right] \quad (9)$$

According to the algorithm described in figure 7, the rule *lex* cannot be applied because of the SK structure condition: input

[5] Identical remarks have been already done in the LFG framework (Wedekind and Kaplan,1993), (Kaplan and Wedekind,1993).

[6] All this information can be collected off-line from the grammar, though in the current implementation it has been done by hand.

```
%% Initial Goal
skgen(Goal,EndString) :-
   has_only_sk(Goal),
   predict_head(Goal,IniString,IntermGoal),
   skhead_corner(Goal,IntermGoal,IniString,EndString).

skgen(Goal,EndString) :-
   has_non_sk(Goal,SubGoals),
   decompose(Goal,SK_structures,NonSK_structures),
   sk_obtain_subgoals(Goal,
                      SK_structures,
                      NonSK_structures,
                      SubGoals),
   generate_top_down(SubGoals,Strings),
   combine(Strings,EndString).

%% Prediction step
predict_head(Goal,IniString,IntermGoal) :-
   lex_entry(Entry,IniString,Syn,Sem),
   syn_link(Goal,Syn),
   sk(Goal,GoalSem),
   unify(GoalSem,Sem).

%% Head Corner Step
skhead_corner(Goal,Goal,String,String).
skhead_corner(Goal,Intermgoal,IniString,EndString) :-
   select_nonsk_rule(IntermGoal,NewGoal,Daughters),
   syn_link(Goal,NewGoal),
   generate_top_down(Daughters,List_Strings),
   combine(IniString,List_Strings,NewString),
   skhead_corner(Goal,NewGoal,NewString,EndString).

%% Top down expansion
generate_top_down([],[]).
generate_top_down([Goal|Rest],EndString) :-
   skgen(Goal,String1),
   generate_top_down(Rest,List_Strings),
   combine(String1,List_Strings,EndString).

%% Specific Information for each grammar. !!
sk_obtain_subgoals((cat:np),
                   SK_structure,
                   NonSK_structure,
                   [(cat:det & sem:SEM1),(cat:n2 & sem:SEM2)]) :-
              select_sk(SK_structure,New_SK_structure,SEM1),
              join(New_SK_strucure,NonSK_structure,SEM2).

sk_obtain_subgoals((cat:n2),
                   SK_structure,
                   NonSK_structure,
                   [(cat:adj & sem:SEM1),(cat:n2 & sem:SEM2)]) :-
              select_nonsk(NonSK_structure,New_nonSK_structure,SEM1),
              join(New_nonSK_structure,SK_structure,SEM2).

...
```

Figure 6: Simplified prolog version of the SKG algorithm

all leaves are labeled with terminals and the tree does not contain any dotted lines *(global-success)*

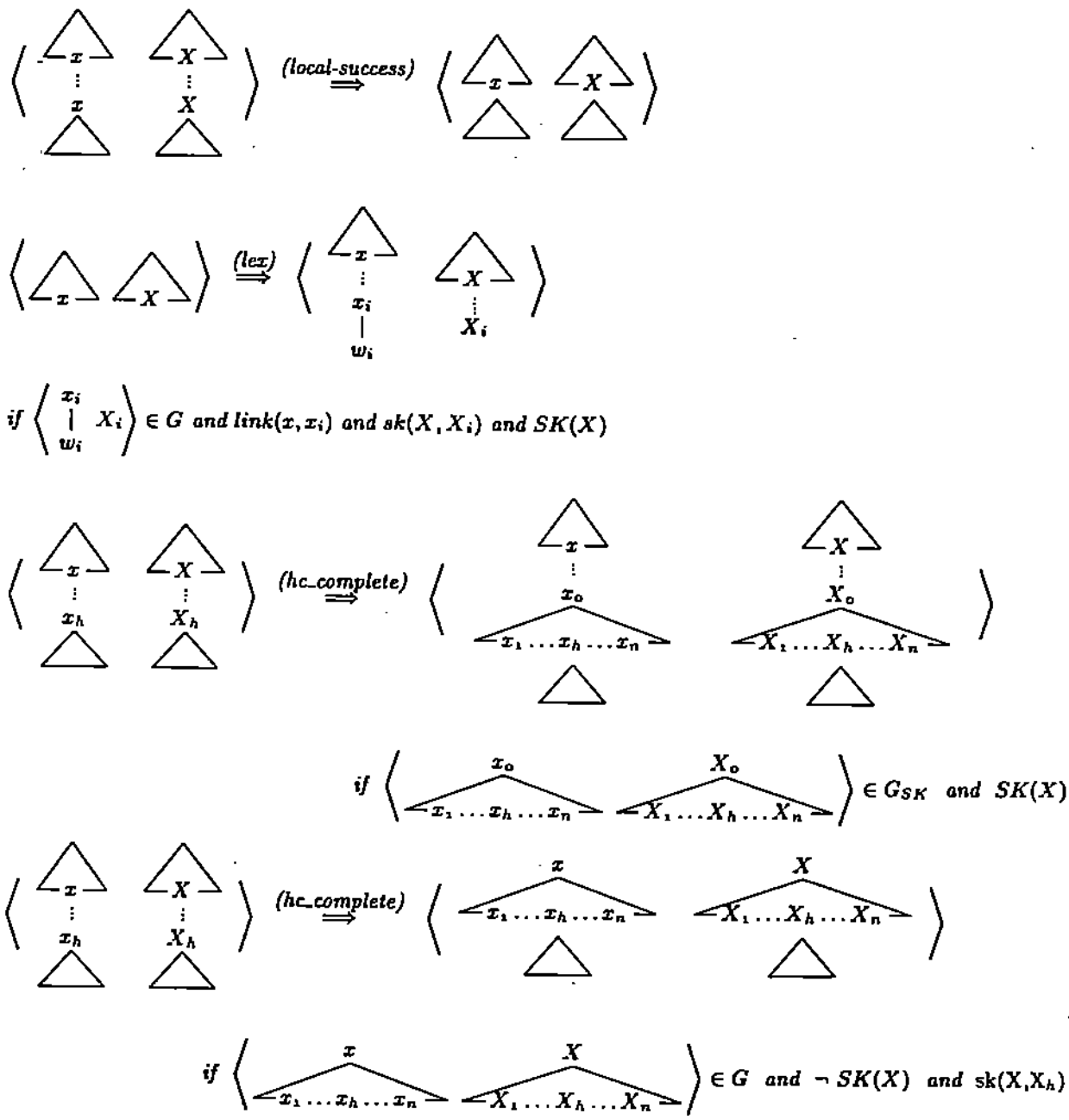


Figure 7: Semantic Kernel Generator (G grammar description; x_i syntactic category; X_i semantic representation; $SK(X)$ is true if X contains only SK information; G_{SK} grammar description with only SK rules; $G_{\neg SK}$ grammar description with only nonSK rules; $sk(X, X_h)$ is true if X_h is SK information of X (Adapted from (König,1994))

semantics does have nonSK information. Accordingly, we can only apply the second *hc_corner step*. This leads us to start from *rule 6* and generate (top-down) the following goals:

$$\begin{bmatrix}\text{cat:} & \text{det} \\ \text{sem:} & [\text{def:} \quad +]\end{bmatrix} \tag{10}$$

$$\begin{bmatrix}\text{cat:} & \text{n2} \\ \text{sem:} & \begin{bmatrix}\text{rel:} & \text{sentence} \\ \text{mods:} & [\text{complex}]\end{bmatrix}\end{bmatrix} \tag{11}$$

Note that the generator has been told about the relation among nonSK information and SK and nonSK rules, so in this case, it knows where the modifiers come from (see rule *sk_obtain_subgoals* in figure 6). The generation of the determiner is straightforward, since it reduces to apply the *lex* rule. In order to generate the *n2* goal (example 11) we proceed as before; this goal has nonSK information, so the generator starts from *rule 8* and generates (top-down) the appropriate subgoals:

$$\begin{bmatrix}\text{cat:} & \text{n2} \\ \text{sem:} & [\text{rel:} \quad \text{sentence}]\end{bmatrix} \tag{12}$$

$$\begin{bmatrix}\text{cat:} & \text{adj} \\ \text{sem:} & [\text{rel:} \quad \text{complex}]\end{bmatrix} \tag{13}$$

Aftger generating the subgoals the generator combines the strings. The generation of the subgoals above is straightforward, since now they do not have nonSK information and the *lex* rule can be applied without problems. The application of the *hc_corner step* to each of the subgoals deserves further comments, since we are running the risk to have *termination* problems. For example, once we have applied the *lex* rule and the *hc_corner* step for *sentence* we obtain the goal in example 12. One may ask whether we could apply rule *8* again and end up having subgoals like the following:

$$\begin{bmatrix}\text{cat:} & \text{n2} \\ \text{sem:} & \begin{bmatrix}\text{rel:} & \text{sentence} \\ \text{mods:} & [\text{X},\ldots]\end{bmatrix}\end{bmatrix} \tag{14}$$

This situation cannot arise, since if we have only SK information only SK rules can be applied, and rule *8* is a nonSK rule (see conditions for the application of the first *hc_complete step* in figure 7).

Another example will clarify how the generator works. Assuming one wants to generate a string for the the semantic representation in figure 4, the following sentences should be generated according to the grammar:[7]

- *the {little,prolog} program generated the complex sentence quickly.*
- *quickly the {little,prolog} program generated the complex sentence.*
- *the {little,prolog} program quickly generated the complex sentence.*

The generator detects that the semantic representation is composed by a SK structure and a Non-SK structure ("quickly"). Thus, according to the grammar, there are several ways of generating these structures given the original goal (which is a sentence). The generator tries the following combinations:

- It generates a string of type "S" for the SK information and a string of type "adv" for the Non-SK information. Both strings can be combined in two ways (which corresponds to rules R1a and R1b). This gives us two of the possibilities.
- It generates a string of type "VP" for the SK information and a string

[7] In this example we will only concentrate in the generation of *quickly* at sentence or vp level; the rest of modifiers (*complex, little, prolog*) for the np level would be generated in an identical manner.

of type "adv" for the Non-SK information (this corresponds to rule R3). After generating these strings, and according to rule R2, we connect the "VP" to "S"

3 Discussion

The main characteristics of the SKG generator are the following:

- It proceeds top-down, generating the appropriate subgoals, when it finds nonSK information.
- It proceeds bottom-up when lexical prediction can be made (when there is only SK information).
- The *head-corner step* for SK information can only be performed using SK rules, thus avoiding *termination* problems.

The benefits and drawbacks of our proposal are clear: the generator has the same processing strategy as the SHDG and it is complete and coherent, but it needs more information about the grammar.

It is interesting to point out that the way our generator works and the distinction between Sk and Non-SK information resembles the definition of the *restrictor* operator and the treatment of modifiers given in (Wedekind and Kaplan,1993), (Kaplan and Wedekind,1993). The difference is obviously the context of application: In (Kaplan and Wedekind,1993) the main interest is structural misalignment between f-structure and semantic representations, whereas our concern is string generation from f-structure like representations.

4 Implementation

Our framework has been the Sicstus-Prolog version of the CUF language[8] plus a layer on top of it which implements the grammar formalism, the (left-corner) parser and the SKG generator.[9]

5 Conclusions

We have shown that for f-structure representations, previously proposed generation algorithms run into problems. However, since the SHDG algorithm has its merits (basic bottom-up strategy, top-down prediction, lexical information available as soon as possible), we have proposed a generation algorithm, the SKG algorithm, that follows the same processing strategy but under other assumptions: For each semantic input it is possible to compute its SK and Non-SK information. We have also shown that our approach resembles the definition of the *restrictor* operator and the treatment of modifiers given in (Kaplan and Wedekind,1993) for dealing with structural misaligments between f-structure and semantic representations. The main drawback of the algorithm is that it needs to collect more information about the grammar.

6 Future work

We are currently investigating how to derive (semi)automatically the SK and nonSK information from the grammar and lexical entries. We also plan to incorporate the ideas presented in (Kay,1996) (chart generation for flat semantic representations) to our algorithm.

7 Acknowledgments

I would like to thank Esther König, Michael Dorna and Nicolas Nicolov for illuminating comments on the ideas presented in this paper. Obviously, I am responsible for any mistake.

[8] CUF is an extension of Prolog by feature terms, types and a sophisticated evaluation strategy (Dörre and Dorna,1993).

[9] Code is available from the author.

References

Stephan Busemann. 1996. The interface between Constraint-Based Grammars and Generation Systems. Draft paper.

Dörre J. and Dorna M.. 1993. CUF - A Formalism for Linguistic Knowledge Representation. Deliverable R.1.2A, DYANA, Institut für Maschinelle Sprachverarbeitung, University of Stuttgart, Germany.

Kaplan, R. and Wedekind, J. 1993. Restriction and Correspondence-based Translation. In *Proceedings of EACL-93*, Utrecht, The Nederlands.

Martin Kay. 1996. Chart Generation. In *Proceedings of ACL-96*

Esther König. 1994. Syntactic-Head Driven Generation. In *Proceedings of Coling-94*, Kyoto, Japan.

Esther König. 1995. LexGram - a practical categorial grammar formalism. In *Proceedings of CLNLP-95*, Edinburgh, Scotland.

Esther König. 1996. Personal communication.

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

Shieber S., van Noord G., Moore R. and Pereira F. 1990. Semantic-head-driven Generation. In *Computational Linguistics*, 16(1)

Wedekind, J. and Kaplan, R.. 1993. Type-Driven Semantic Interpretation of f-structures. In *Proceedings of EACL-93*, Utrecht, The Nederlands.