Generating from a Deep Structure *

Claire Gardent
Université Blaise Pascal - Clermont Ferrand (France)
and University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW (Scotland)

Agnes Plainfossé
Laboratoires de Marcoussis
Route de Nozay
91460 Marcoussis (France)

ABSTRACT: Noncanonical semantic representations are representations which cannot be derived by some grammar G although they are semantically equivalent to representations which can be derived by G. This paper presents a generation algorithm which deals with non-canonical input. The proposed approach also enhances portability and language independence in that (i) linguistic decisions made by independent modules (e.g., planner, transfer component) can be communicated to the generator in a natural way and (ii) the same algorithm coupled with different grammars will yield sentences in the corresponding languages.

1 Introduction

Two major requirements on a generator is that it be correct and complete. A generator is said to be correct if given two semantic representations *R1* and *R2* which are not semantically equivalent, *R1* and *R2* do not generate the same string. A generator is said to be complete if any two semantically equivalent representations generate the same set of strings.

An obvious case of incompleteness occurs when the generator fails to terminate on some well-defined input. Another less obvious cause for incompleteness can be explained as follows. Consider a grammar G and its associated semantic representation language L. It is often the case that syntactically different strings of L will have equivalent semantics. A simple case in point is the semantic equivalence holding between $\phi \wedge \psi$ and $\psi \wedge \phi$ in e.g. propositional logic. On the other hand, it is also often the case that the same grammar G will not derive for a given string all the formulae which may represent its meaning. From the point of view of generation, this means that given two semantically equivalent representations *R1* and *R2* there is always a possibility that *R1* generates a string *S* but that *R2* doesn't because *R2* is not derivable under G. Representations that cannot be derived under a grammar G are said to be ***noncanonical*** with respect to G.

In this paper, we present a generation algorithm for Unification Categorial Grammar [6] (UCG) which addresses the problem raised by non-canonical input (section 2 and 3). An interesting upshot of the strategy we propose is that it allows for language independent generation. Section 4 illustrates this point by considering how, starting from one semantic representation, two sentences can be generated: one in English and one in French. Section 5 relates our work to previous proposals by Van Noord and Shieber.

2 Generating from a deep structure

It is sometimes the case that a grammar will assign to a string several possible derivations with equivalent semantics. This phenomena is particularly acute in categorial grammars [2] and is referred to in the literature as that of ***spurious ambiguity.*** In grammars where the semantics is built by unification, the syntactic differences holding between these equivalent semantics resides in the relative ordering of the subformulae within the formula. That is, there is a direct relationship between the syntactic shape of a semantic formula and the derivational history of the corresponding string. Consequently, a given formula will be non-canonical wrt to a particular grammar G if the relative sequencing of its subformulae does not reflect a possible derivation in G. Hence, to allow for generation from non-canonical input, we need to abstract away from the derivational information reflected in the linear ordering of the input formula. Three major alternatives come to mind. First, we could try to generate all sentences whose semantics are logically equivalent to the input semantics. In UCG, this means that generation is carried out with the two additional logical axioms of associativity and commutativity. However, this solution produces a search space factorial in the number of conjunctions and must thus be rejected as computationally intractable.

*The work reported here has been carried out as part of the ESPRIT project P393 ACORD on "The Construction and Interrogation of Knowledge-Bases using Natural Language Text and Graphics". It is the result of joint work with Michael Reape of the Center for Cognitive Science, University of Edinburgh (Scotland, UK).

The second possibility is to define a semantic representation language for which all well-formed formulas are in normal form. This approach is essentially unavailable to any grammar framework in which the semantics of a given expression results from the unification of partially specified semantic representations because normal forms can only be defined on languages with fully instantiated formulae.

A third possibility consists in generating from an alternative representation i.e. one that is related to but not identical with the semantic representation used by the grammar. This is what we chose to do. The alternative representation we adopted is closely related to *D-structure* in GB theory where *D-structure* is a level of syntactic structure which mirrors semantic functor-argument dependencies. Syntactic information is encoded in terms of schematic $\overline{X}$ theory familiar from modern generative grammar. The deep structures (DS) we generate from consist of four types: *heads, complements, modifiers* and *specifiers* (we follow LFG *f*-structure and UCG subcategorisation structure in treating subjects as ordinary complements rather than specifiers of clauses) whereby *Specifiers* are of the form: `specifier(Semantics, Head)`. That is, they specify their own semantics and the properties of their head. In contrast, *Heads* are of the form: `head(Semantics, ArgList, AdjunctList)`. That is, they specify their own head semantics and a list of arguments and adjuncts which are also either specifier or head structures. All of these structures also allow the encoding of syntactic requirements on arguments and adjuncts.

In relation with the problem raised by non-canonical input, an important property of DSs is that they contain no indication of either surface syntactic order of the complements and adjuncts or of the relative scope of quantifiers occurring in either complements or modifiers. Instead, thematic dependencies between subformulae are kept track of by the $\overline{X}$ schema where no reference is made to derivational history. The generator is thus free to realize both scope and surface syntactic structure in any way which is consistent with the deep structure specification and the particular grammar used. The reader might object to this elimination of scope distinctions. However, within UCG any scope distinctions which are produced by the individual grammars or as a result of some semantics construction process are in fact artefactual. Furthermore, it might reasonably be argued that it should be possible to generate all possible scopes. This is typically done with *quantifier shifting* rules. Our solution is simply not to specify scope.

An immediate consequence of using DSs is that non-canonical input is no longer a problem. The reason for this simply is that the generation algorithm no longer relies on the assumption that the input semantic representation is canonical i.e. derivable under the grammar used. Rather, the assumption is that the input will be some well-formed DS that will contain all the information contained in the corresponding semantics but none of the information embodied in the linear ordering of the formula about the derivational history of the corresponding string.

The use of DSs has two other consequences. First, by allowing for the association of syntactic with semantic information, *D-structures* offer a way to mediate the results of linguistic decisions made by an eventual planner to the generator. This may be useful. For instance, NP planning could be accounted for. In the present context, a planner is any system which given some information about *what* to say will return some decision about *how* to say it. For instance, if we want to express the fact that Jon runs, the planner will have to decide on *how* to refer to Jon, i.e. it could decide to describe him using a complex NP as in *'the man with the red scarf who stands next to Irene'*, or a pronoun e.g. *'he'* or simply his name i.e. *'Jon'*. The point is that the syntactic decision made by the planner must be communicated to the generator. Since DSs contain syntactic information, they are a good candidate for the necessary interface between planner and generator.

A second advantage of DSs is that because they are language independent, they allow for language independent generation. That is, for any acceptable input deep structure, the algorithm presented below will generate e.g., a French sentence if coupled with a UCG grammar for French and an English sentence if coupled with a UCG grammar for English. This is only possible because the input deep structure the generation algorithm relies on is both sufficiently abstract to be language-independent and general enough that it can be mapped onto language dependent surface syntactic structures. Language independent generation is discussed in more detail in section 4.

3 The basic algorithm

3.1 A brief introduction to UCG

In UCG the basic linguistic unit is a *sign* which includes phonological, syntactic, semantic and ordering information. In the sequel, a sign will be represented either by a complex feature structure or as `Pho:Synt:Sem:Order`. The phonological field of a sign contains its orthographic string. The syntactic field is categorial i.e. it can be either basic (e.g s,np,n etc) or complex in which case, it will be of the form $C/Sign$ where C is a syntactic field and $Sign$ is a sign. Moreover, any basic category can be assigned some morphosyntactic information. For instance $s[fin]$ denotes the category *sentence* with morphology feature value: *finite*. The semantic field contains the semantics of the expression whereby the semantic representation language is a linear version of Discourse Representation Theory in which each condition is preceded by a sorted variable called the index. As in most unification based grammars, the semantics of any expression results from the unification of the semantics of its subparts. Finally, the Order field is a binary feature with value either *pre* or *post* which constrains the applicability of grammar rules.

Grammar rules in UCG are of two types: binary and unary. Binary rules include forward and backward functional application. These are stated below.

```
Pho:(Synt/Sign):Sem:Order, Sign
                --> Pho:Synt:Sem:Order
     if the order value of Sign is pre

Sign, Pho:(Synt/Sign):Sem:Order
                 --> Pho:Synt:Sem:Order
     if the order value of Sign is post
```

Unary rules are of the form $\alpha \rightarrow \beta$ where α and β are signs. Unary rules are used for the treatment of unbounded dependencies, syntactic forms of type-raising and subcategorisation for optional modifiers.

3.2 A sketch of the algorithm

Following work by [1], [5] and [3], the algorithm we present here follows a mixed top-down and bottom-up strategy.

The generation process starts with a deep structure DS and a sign Sign whose syntax embodies the goal category (e.g. sentence(finite)). **get_deepstr_info** extracts from the deep structure some semantic (Sem) and syntactic (Synt) information on the next sign to be generated. **create_sign** creates a new sign Sign0 on the basis of Sem and Synt. Lexical look-up on Sign0 returns a sign with instantiated syntax and phonology. The call to **reduce** ensures that this lexical sign is reduced to the goal sign Sign in the process instantiating the generated string.

```
generate(DS, Sign) :-
  get_deepstr_info(DS,[Synt,Sem],RestOfDS),
  create_sign(Synt,Sem,Sign0),
  lexical(Sign0),
  reduce(Sign0,Sign,RestOfDS).
```

There are two main ways of reducing a sign Sign0 to a goalsign Sign. The base case occurs when Sign0 unifies with Sign and the deep-structure is empty i.e. all the input semantic material has been made use of in generating the result string. The recursive case occurs when Sign0 is a syntactic functor. If the syntax of Sign0 is of the form Result/Active, we apply Result/Active to Active thus getting a new sign Result. **retrieve** non-deterministically retrieves from the current deep structure DS, a substructure SubDS and returns the remaining deep-structure NewDS. The argument Active is then generated on the basis of the extracted sub-structure SubDS with a new goal sign whose syntax is that predicted by the syntactic functor Sign0. The resulting sign Result is recursively reduced to the original goal sign Sign.

```
reduce(Sign,Sign,[[],[]]).
reduce(Sign0,Sign, DS) :-
  active(Sign0,Active),
  apply(Sign0,Active,Result),
  retrieve(DS,SubDS,NewDS),
  generate(SubDS, Active),
  reduce(Result,Sign,NewDS).
```

The algorithm presented above makes many simplifying assumptions which are incompatible with a wide coverage UCG grammar. To produce a complete generator with respect to UCG we need to extend the basic algorithm to account for type-raised NPs, identity semantic functors, lexical modifiers and unary rules. For more details on the general content of these extensions see [1]. For their implementation cf. the listing of the generation algorithm given in the appendix.

4 Bilingual Generation

Consider the following synonymous sentences.

a The mouse misses the cat
b Le chat manque à la souris
(Lit. *the cat misses to the mouse*)
(1)

There are two main differences between (1a) and (1b). First, a NP (*the mouse*) translates to a PP (*à la souris*). Second, a structural transfer occurs i.e. the object NP in (1a) becomes a subject in (1b) and vice-versa. For the generator described above, this poses no particular problem. Because DSs encode thematic rather than grammatical dependencies, structural transfer is no issue. Further, since at DS all arguments are represented as NPs [1], the generation of (1a) is straightforward. Generating (1b) is a little more intricate but results naturally from the interaction of the generator with the grammar [2]. Note that if the PP were represented as such in the DS, then generation would fail for the English sentence. This suggests that the deep structures we generate from offer the right level of abstraction for generation to be possible in several languages.

The case of structural transfer illustrated in (1) is a good example of the problems that occur with generators that are unable to deal with non-canonical input. To illustrate this consider the following situation. Suppose that given two grammars, one for English(G_E) and one for French (G_F), (1a) and (1b) each have one unique derivation with resulting semantics as in (2).

a The(mouse(m), the(cat(c), miss(m,c))
b The(cat(c), the(mouse(m), miss(m,c)))
(2)

Furthermore, suppose (3a) is non-canonical with respect to G_F (i.e. (3a) is not derivable under G_F) and (3b) is non-canonical wrt G_E. For any generator G that cannot deal with non-canonical input, this means that G cannot be used in a system where parsing occurs on one language

[1]This is in accordance with the view that prepositions occuring within argumental PPs have no semantic content.

[2]For more details on the generation of subcategorised PPs within UCG see [1].

and generation on another. More to the point, if G is coupled with the grammar G_E, then it will fail to generate when given (2b) as input - and similarly when coupled with G_F and given input (2a). To understand why deep structures allow for grammar independent generation, let us first examine why traditional top-down/bottom-up generators such as the one described in [1] fail on non-canonical input. [3] Consider the case where we try to generate under G_E the English sentence in (1a) from the semantic (2b) and - as already mentioned - (2b) is non-canonical wrt G_E. The main steps of the generation process will be as follows. [4] Suppose the goal sign is *Sign*0 with category $s[fin]$. First, a sign *Sign*1 is created whose semantics is as in (2b). Lexical access on *Sign*1 returns the sign for *'the'*. On the basis of the syntactic and semantic predictions made by *Sign*1, the sign *Sign*2 for *'cat'* is then generated. Reduction of *Sign*1 with *Sign*2 yields a new sign *Sign*3 with phonology *'the cat'* and syntax $C/(C/np)$ [5]. In turn, *Sign*3 makes some predictions which lead to the generation of a new sign *Sign*4 with syntax $C/(C/np)$ and phonology *'the mouse'*. Finally, on the basis of *Sign*4, the sign *Sign*5 for *'miss'* is generated. At this point in generating, the two signs in (3) must combine to reduce to a sign with category C/np.

$$\left[\begin{array}{ll}
\multicolumn{2}{l}{\mathtt{pho : Wb}} \\
\mathtt{synt : C/} & \left[\begin{array}{ll}
\multicolumn{2}{l}{\mathtt{pho : Wa}} \\
\mathtt{synt : C/} & \left[\begin{array}{l}
\mathtt{pho : themouse} \\
\mathtt{synt : np} \\
\mathtt{sem : m.mouse(m)} \\
\mathtt{order : 01}
\end{array}\right] \\
\multicolumn{2}{l}{\mathtt{sem : VP}} \\
\multicolumn{2}{l}{\mathtt{order : 01}}
\end{array}\right] \\
\multicolumn{2}{l}{\mathtt{sem : the(m.mouse(m), VP)}} \\
\multicolumn{2}{l}{\mathtt{order : 02}}
\end{array}\right]$$

$$\left[\begin{array}{ll}
\multicolumn{2}{l}{\mathtt{pho : misses}} \\
\mathtt{synt : s/} & \left[\begin{array}{l}
\mathtt{pho : Wa} \\
\mathtt{synt : np[nom]} \\
\mathtt{sem : m.NP1} \\
\mathtt{order : pre}
\end{array}\right] / \left[\begin{array}{l}
\mathtt{pho : Wb} \\
\mathtt{synt : np[acc]} \\
\mathtt{sem : c.NP2} \\
\mathtt{order : post}
\end{array}\right] \\
\multicolumn{2}{l}{\mathtt{sem : m.miss(m,c)}} \\
\multicolumn{2}{l}{\mathtt{order : 03}}
\end{array}\right]$$

But under the UCG rules of combination (see 3.1), these two signs cannot combine because of the unification clash occuring between the semantics of the accusative NP in the verbal sign (c.NP2) and that of the NP sign within the sign for *'the mouse'* (m.mouse(m)). Hence generation fails. Consider now how the problem is dealt with when generating from deep structures. Rather than being as indicated in (2b), the input to the generator is [6]

$$\begin{array}{l}
head(miss(m,c), \\
[specifier(the, head(mouse(m), [], [])), \\
specifier(the, head(cat(c), [], []))] \\
[])
\end{array} \tag{3}$$

Roughly, generation will proceed as follows. Suppose the goal sign *Sign*0 has category $s[fin]$. First, the semantics corresponding to the head of the clause (i.e. $miss(m,c)$) is extracted from (3) and a sign *Sign*1 is created with semantics $miss(m,c)$. Lexical access on *Sign*1 returns the sign given in (3) above. *Sign*1 must then be reduced to *Sign*0 with category $s[fin]$. At this stage, the remaining DS is $[specifier(the, head(mouse(m), [], [])), specifier(the, head(cat(c), [], []))]$ To generate the first argument
of *Sign*1, we then have the choice between generating on the basis of $specifier(the,\ head(mouse(m), [], []))$ or of $specifier(the,\ head(cat(c), [], []))$ [7] As demonstrated above, if we generate the sign for *'the mouse'* first, reduction cannot apply and generation will fail. But here, failure is only temporary and on backtracking, the sign for *'the cat'* will eventually be generated; it will then reduce with *Sign*1 to generate *Sign*2 with phonology *'misses the cat'*. At this point, the remaining DS will be $[specifier(the, head(mouse(m), [], []))]$. This will trigger the generation of *Sign*3 with phonology *'the mouse'* which will then combine with *Sign*2 to reduce to *Sign*0 with resulting phonology *'the mouse misses the cat'*.

To generate the French sentence *'le chat manque à la souris'*, the same generation process applies but this time in connection with G_F and in a reverse order i.e. the sign for *'la souris'* (the mouse) is generated before the sign corresponding to the NP *'le chat'* (the cat). Further, because in the French lexicon *'manque'* (miss) subcategorises for a dative NP, the preposition *à* is generated and combined with the sign for *'la souris'* before reduction of the thus obtained PP with the verb. Because DSs make no assumption about the linear ordering of the constituents to be generated, the problem raised by non-canonicity simply does not arise.

5 Comparisons with Related Research

To compare our algorithm with previous work, we first show how it can be amended to phrase structure grammars. Consider the following extension to **reduce**.

```
reduce(Sign0, Sign, DS) :-
  rule(Mom, Sign0, Kids),
```

[3]Note that in this case, reduction to normal form is no longer a possible solution even if we were able to define a normal form for our semantic representation language. For suppose that (2a) is the normal form, then (1b) is not derivable and if (2b) is, then (1a) is not derivable.

[4]For more information on the details of the generation procedure, see [1].

[5]For the sake of clarity, the syntactic part of *Sign*3 is here simplified in that non-syntactic fields (Phonology, Semantics etc.) are omitted. Note also that in UCG, NPs are typeraised i.e they are assigned the syntactic category $C/(C/np)$ as opposed to just np.

[6]For the sake of simplicity, the syntactic information usually contained in the deep structures input to the generator is here omitted.

[7]cf. the non-determinism of the **retrieve** predicate.

```
  generate_sisters(Kids, DS, NewDS),
  reduce(Mom, Sign, NewDS).

generate_sisters([], DS, DS).
generate_sisters([H|T], DS, NewDS) :-
  index(H,Idx),
  match(Idx,DS,SubDS,NewDS1),
  generate(SubDS, H),
  generate_sisters(T, NewDS1, NewDS).
```

This clause is very similar in structure to the second clause of reduce, the main difference being that the new clause makes fewer assumptions about the feature structures being manipulated. rule enumerates rules of the grammar, its first argument representing the mother constituent, its second the head daughter and its third a list of non-head daughters which are to be recursively generated by the predicate generate_sisters. The behaviour of this clause is just like that of the clause for reduce which implements the UCG rules of function application. On the basis of the generated lexical sign Sign0 an application of the rule is hypothesised and we then attempt to prove that rule application will lead to a new sign Mom which reduces to the original goal Sign.

Having generalised our basic algorithm to phrase structure grammars, we can now compare it to previous work by [5] and [3]

Van Noord's Bottom-Up Generator (BUG) is very similar in structure to our basic algorithm. Closer examination of the two programs however reveals two differences. The first is that daugthers in a rule are separated into those that precede the semantic head and those that follow it. The second more meaningful difference involves the use of a 'link' predicate implementing the transitive closure of the semantic head relation over the grammar rules. The *link* predicate is similar in purpose to reachibility tables in parsing algorithms and contributes to reducing the search space by producing some syntactic information on the sign to be generated. However, such a predicate is of little use when generating with a categorial grammar in particular and with any strongly lexicalist linguistic theory in general since in these, the grammar rules are extremely schematised. Their information content is so impoverished that the computation of and resort to a link predicate cannot be expected to reduce the search space in any meaningful way. In the algorithm presented above however, this shortcoming is redressed by exploiting the syntactic information contained in the deep-structure we start from.

In [5], Shieber et al. present a "semantic-head-driven" generation algorithm that is closely related to van Noord's. In contrast to Van Noord's algorithm however, this algorithm also operate on grammars violating the *semantic head constraint* (SHC) according to which any semantic representation is a further instantiation of the semantic representation of one of its constituents called the semantic head. This is achieved as follows. First, a distinction is made between chain-rules and non-chain-rules whereby non-chain-rules are used to introduce semantic material syncategorematically. The distinction between the two types of rules can be sketched as follows.

```
1. Chain-rule (Sem, lhs --> Head(Sem), Sisters)

2. Non-Chain-rule (Sem, lhs(Sem) --> Daughters)
```

(1) indicates that given a semantic *Sem*, a chain rule will be such that *Sem* unifies with the head daughter's semantics whilst (2) shows that non-chain-rules are such that the input semantics must unify with the semantics of the lhs of the rule. The intuition is that non-chain-rules will help find the lowest node in the derivation tree whose semantics unify with the input semantics. Furthermore, the top-down base case for non-chain-rules corresponds to the case in which the lhs of the rule has no non-terminal daughters i.e. to lexical look up. Consider now the top call to generate.

```
generate(Root) :-
  non_chain_rule(Root,Pivot,Rhs),
  generate_rhs(Rhs),
  connect(Pivot,Root).
```

Two cases obtain with regard to the application of the non- chain-rule predicate. Either the base case occurs and lexical look-up takes place exactly as in our algorithm or a non-chain-rule is triggered top-down before the constituents in the rhs are generated by a recursive call to generate. Hence the solution to the introduction of syncategorematic material is essentially a reintroduction of the top-down generation strategy. The result is that there is no guarantee that the algorithm will terminate. This point seems to have been overlooked in [5]. Therefore, the extension may be of less utility than it appears to be at first sight although it may well be the case for linguistically motivated grammars that termination problems never arise.

6 Further Research

The general backtracking regime characterising the algorithm presented in this paper means that failure at a first attempt to generate might induce the recomputation of partial results. Perhaps the use of a chart could contribute to enhance generation efficiency. In relation to [4] where chart edges contain no ordering information, it would be interesting to investigate whether during the generation process some ordering information can be recovered. That is, whether the chart could be constructed in such a way that the relative positioning of edges mirrors the knowledge embodied in the grammar about linear precedence within and between constituents. In this way, only the relevant part of the chart would need to be looked up before attempting to build a new edge.

The algorithm described above is implemented in CProlog on a Sun4 and constitutes part of the generation component in the ACORD prototype. The generator can be coupled with either a UCG grammar for French or one for English thus generating either French or English sentences.

References

[1] **Calder, J., Reape, M. and Zeevat, H. [1989]** An Algorithm for Generation in Unification Categorial Grammar. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, University of Manchester Institute of Science and Technology, Manchester, England, 10-12 April, 1989, 233-240.

[2] **Gardent, C., Bes, G., Jurie,P.F. and Baschung, K. [1989]** Efficient Parsing for French. In *Proceedings of the 27th annual meeting of the Association for Computational Linguistics*, University of British Columbia. Vancouver, 26-29 June 1989, 280-287.

[3] **van Noord, G. [1989]** BUG: A Directed Bottom Up Generator for Unification Based Formalisms. Manuscript. Department of Linguistics, University of Utrecht, March 14, 1989.

[4] **Shieber, S. [1988]** A Uniform Architecture for Parsing and Generation. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, 22-27 August, 1988, 614-619.

[5] **Shieber, S., van Noord, G., Moore, R. and Pereira, F.C.N. [1989]** A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms. In *Proceedings of the* **27th** *Annual Meeting of the Association for Computational Linguistics.* University of British Columbia, Vancouver, British Columbia, Canada, 26-29 June, 1989, 7-17.

[6] **Zeevat H., Klein, E. and Calder, J. [1987]** An Introduction to Unification Categorial Grammar. In Haddock, N.J., Klein, E. and Morrill, G. (eds.) *Edinburgh Working Papers in Cognitive Science, Volume 1: Categorial Grammar, Unification Grammar and Parsing.*

Listing of the whole program
(Low level procedures have been omitted)

```
generate(DeepStr, Sign) :-

get_deepstr_info(DeepStr,[Synt,Sem],RestOfDeepStr),
  create_sign(Synt,Sem,Sign0),
  lexical(Sign0),
  reduce(Sign0,Sign,RestOfDeepStr).

reduce(Sign,Sign,[[],[]]).
reduce(Sign0,Sign, DeepStr) :-
  active(Sign0,Active),
  apply(Sign0,Active,Result),
  retrieve(DeepStr,SubDeepStr,NewDeepStr),
  generate(SubDeepStr, Active),
  reduce(Result,Sign,NewDeepStr).

reduce(Sign0, Sign, DeepStr) :-
   transform(Sign0, Sign1, DeepStr, NewDeepStr),
   reduce(Sign1, Sign, NewDeepStr).


% Identity Semantic Functors

transform(Sign,NewSign,DeepStr,DeepStr) :-
        not_idsign(Sign),
        create_id_functor(IdSemFctor, Sign),
        identity(IdSemFctor),
        apply(NewSign,IdSemFctor,Sign),
        defreeze_order(AdjSign, Sign, NewSign).

% Lexical Adjuncts

transform(Sign,NewSign,DS,NewDS) :-
        create_lexical_adjunct(Sign,
ASign,DS,NewDS,DS2),
        generer(DS2, ASign),
        apply(NewSign, ASign, Sign).

% Type-raise Verbs to C/(C/NP)

transform(Sign,NewSign,DS,NewDS) :-
        type_raise_vb_to_np(Sign, RaisedSign),
        get_sub_deepstr(Sign, DS, SubDS, NewDS),
        generer(SubDS, RaisedSign),
        apply(NewSign, RaisedSign, Sign).



% Unary rules

transform(Sign,NewSign,DeepStr,DeepStr) :-
        unary_rule(NewSign,Sign).

% Identity Semantic Functor
% (Case marking Prepositions)

transform(Sign,NewSign,DeepStr,DeepStr) :-
        active(Sign,VB),
        active(VB, NP),
        category(NP, np),
        create_id_prep(Np,PREP),
        identity(PREP),
        apply(NewSign, Sign, PREP).
```