Self–Monitoring with Reversible Grammars

Günter Neumann* and Gertjan van Noord,†

*Deutsches Forschungszentrum
für Künstliche Intelligenz
Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11
neumann@dfki.uni-sb.de

†Rijksuniversiteit Groningen
Postbus 716
NL 9700 AS Groningen
vannoord@let.rug.nl

Abstract

We describe a method and its implementation for self-monitoring during natural language generation. In situations of communication where the generation of ambiguous utterances should be avoided our method is able to compute an unambiguous utterance for a given semantic input. The proposed method is based on a very strict integration of parsing and generation. During the monitored generation step, a previously generated (possibly) ambiguous utterance is parsed and the obtained alternative derivation trees are used as a 'guide' for re-generating the utterance. To achieve such an integrated approach the underlying grammar must be *reversible*.

1 Introduction

In many situations of communication a speaker need not to worry about the possible ambiguity of what she is saying because she can assume that the hearer will be able to disambiguate the utterance by means of contextual information or would otherwise ask for clarification. But in some situations it is necessary to avoid the risk of generating ambiguous utterances that could lead to misunderstanding by the hearer, e.g., during the process of writing text, where no interaction is possible, or when utterances refer to actions that have to be performed directly or in some specific dialog situations (e.g. having an interview with a company).

The need to generate un-ambiguous utterances is also relevant for the development of natural language generation systems. For example in the case of an intelligent help-system that supports the use of an operating system (Wilensky *et al.*, 1984), asking an inexperienced user to 'Remove the folder with the system tools' could have tremendous effects on the system itself.

If one assumes a modular division of the natural language generation task between two stages of the language production process — deciding what to say (*conceptual* level) and deciding how to say it (*grammatical* level) — it is not realistic to expect that the conceptual component will be able to specify the input for the grammatical component such that ambiguous utterances can be avoided.

If it were possible to specify the input in such a way, then this would mean that the conceptual component has to provide all information needed by the grammatical component to make decisions about lexical and syntactic choices. Hence, the conceptual component would need detailed information about the language to use. But this would blur the distinction between the grammatical and the conceptual level, because this would imply that both components share the grammar (see also Appelt (1989), Meteer (1990), Neumann (1991)).[1]

In order to maintain a modular design additional mechanisms are necessary to perform some *monitoring* of the generator's output. Several authors argue for such additional mechanisms (Jameson and Wahlster, 1982; De Smedt and Kempen, 1987; Joshi, 1987; Levelt, 1989). For example, Levelt (1989) pointed out that "speakers monitor what they are saying and how they are saying it". In particular he shows that a speaker is also able to note that what she is saying involves a potential ambiguity for the hearer and can handle this problem by means of self-monitoring.

In this paper we describe an approach for self-monitoring which allows to generate unambiguous utterances in such situations where possible misunderstandings by the user have to be avoided. The proposed method is based on a very strict integration of parsing and generation. During self-monitoring a generated ambigu-

[1] As pointed out in Fodor (1983) one of the characteristic properties of a module is that it is *computationally autonomous*. But a relevant consideration of computationally autonomy is that modules do not share sources (in our case the grammar).

ous utterance is parsed and the obtained alternative derivation trees are used as a 'guide' for the 'monitored' generation step. We will show that such an integrated approach makes only sense with reversible grammars. To our knowledge, there is at present no algorithm that solves the problem of generating un-ambiguous utterances by means of self-monitoring.

2 Overview of the Monitoring Algorithm

Our approach is based on a *strict integration of parsing and generation* in that parsing is used to detect whether a produced utterance is ambiguous or not. The advantages of using comprehension mechanisms to facilitate monitoring are for example mentioned in Levelt (1989). In his model parsing and generation are performed in an isolated way by means of two different grammars. The problem with this view is that generation of un-ambiguous paraphrases can be very inefficient, because the source of the ambiguous utterance is not used to guide the generation process.

To overcome this problem the basic idea of our approach is to operate with *derivation trees* obtained during the generation and parsing step. In short, the algorithm works as follows. Firstly, it is checked whether a produced utterance S of an input form LF is ambiguous, by parsing S. If during parsing e.g. two readings LF and LF′ are deduced LF is generated again along the parse trees obtained for S. Now an utterance S′ can be generated that has the same meaning but differs with respect to the ambiguity source of S.

In this way the derivation trees obtained during parsing of a previously generated utterances are used as a *guide* during monitored generation. Grammatical structures obtained during parsing are used directly to restrict the search space during generation. At this point it should be clear that the only way in order to be able to generate 'along parsed structures' is to use *reversible* grammars. This ensures that every sentence produced by the generator can be parsed. Similarly, for every semantic structure computed by the parser, the generator delivers an utterance.

3 A monitoring strategy

A naive strategy. The first and most straightforward solution to obtain only un-ambiguous utterances during generation could be described as a 'brute force' solution. The generator derives possible utterances for a given logical form. For each of these utterances it is easy to check whether it is ambiguous or not, by counting the results the parser delivers for that utterance.

In a Prolog implementation this simple solution can be defined as follows. Note that we assume for simplicity that linguistic signs are represented with terms `sign(LF,Str,Syn,Der)` where `LF` represents the semantic information, `Str` represents the string and `Syn` represents syntactic information. The fourth argument position will be used later in this paper to represent *derivation trees*.

```
monitor(sign(LF,Str,Syn,Der)):-
    generate(sign(LF,Str,Syn,Der)),
    unambiguous(Str).

unambiguous(Str) :-
    setof(LF,D^S^parse(sign(LF,Str,S,D),
          [El]).
```

The predicates `parse/1` and `generate/1` call respectively the underlying parser and generator. The `setof` predicate is used to obtain the set of solutions of the `parse` predicate, where `^` indicates that `D` and `S` are existentially quantified. By instantiating the resulting set as a set with exactly one element, we implement the idea that the utterance should be un-ambiguous (in that case there is only one parse result). Given Prolog's search strategy this definition implies that the generator generates solutions until an unambiguous utterance is generated.

The problem with this 'generate and test' approach is that the search of the generator is not directed by the goal to produce an un-ambiguous result. We will now present a more involved monitoring strategy which is oriented towards the goal of producing an un-ambiguous utterance.

Ambiguities are often 'local'. A fundamental assumption is that it is often possible to obtain an un-ambiguous utterance by slightly changing an ambiguous one. Thus, after generating an ambiguous utterance, it may be possible to change that utterance *locally*, to obtain an un-ambiguous utterance with the same meaning. In the case of a simple lexical ambiguity this idea is easily illustrated. Given the two meanings of the word 'bank' ('river bank' vs. 'money institution') a generator may produce, as a first possibility, the following sentence in the case of the first reading of 'bank'.

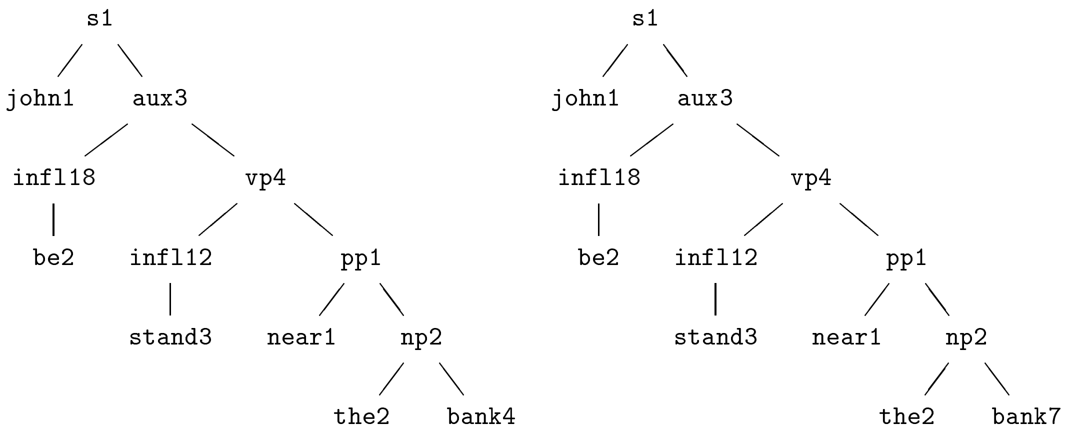

Figure 1: Derivation trees

(1) John was standing near the bank while Mary tried to make a picture of him.

To 'repair' this utterance we simply alter the word 'bank' into the word 'river bank' and we obtain an un-ambiguous result. Similar examples can be constructed for structural ambiguities. Consider the German sentence:

(2) Heute ist durch das Außenministerium bekanntgegeben worden, daß Minister van den Broek den jugoslawischen Delegationsleiter aufgefordert hat, die Armee aus Kroatien zurückzuziehen.
Today it was announced by the ministry of foreign affairs that minister van den Broek has requested the Yugoslav delegation leaders to withdraw the army from Croatia.

which is ambiguous (in German) between 'withdraw [the army of Croatia]' and '[withdraw [the army] away from Croatia]'. In German this ambiguity can be repaired *locally* simply by changing the order of 'aus Kroatien' and 'die Armee', which forces the second reading. Thus again we only need to change only a small part of the utterance in order for it to be un-ambiguous.

Locating ambiguity with derivation trees. We hypothesise that a good way to characterise the location of the ambiguity of an utterance is by referring to the notion of 'derivation tree'. We are assuming that the underlying grammar formalism comes with a notion 'derivation tree' which represents how a certain derivation is licenced by the rules and lexical entries of the grammar. Note that such a derivation tree does not necessarily reflect how the parser or generator goes about *finding* such a derivation tree for a given string or logical form. For example, the derivation trees of the two readings of 'john is standing near the bank' may look as in figure 1. The intuition that the ambiguity of this sentence is local is reflected in these derivation trees: the trees are identical up to the difference between `bank4` and `bank7`. In our examples each sign `sign(LF,Str,Syn,D)` is specified for its corresponding derivation tree `D`. In Prolog such a tree is represented with terms of the form `t(Label,Ds,M)` where `Label` is the node name (the unique name of a rule) and `Ds` is a list of Daughter trees. The third argument position will be explained below.

Given a derivation tree t of a generated sentence s, we mark the places where the ambiguity occurs as follows. If s is ambiguous it can be parsed in several ways, giving rise to a set of derivation trees $T = t_1 \ldots t_n$. We now compare t with the set of trees T in a *top-down* fashion. If for a given node label in t there are several possible labels at the corresponding nodes in T then we have found an ambiguous spot, and the corresponding node in t is *marked*. Thus, in the previous example of structural ambiguity we may first generate sentence (2) above. After checking whether this sentence is ambiguous we obtain, as a result, the marked derivation tree of that sentence. A marked node in such a tree relates to an ambiguity. The relevant part of the resulting derivation tree of the example above may be the tree in figure 2.

Marking a derivation tree. The predicate `mark(Tree,Set)` marks the generated tree `Tree` given the trees `Set` found by the parser. The third argument `M` of the terms `t(Label,Ds,M)` repre-

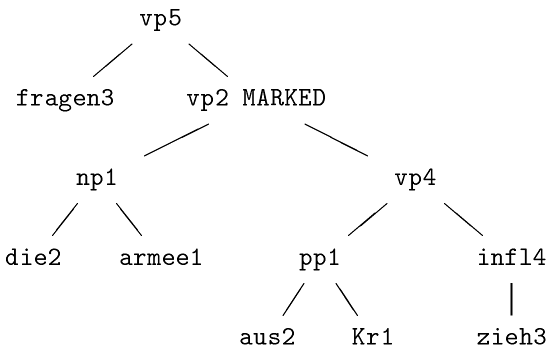

Figure 2: Marked derivation tree

senting derivation trees indicates whether the current node is marked (in that case the value is y) or not (using the value n). Subtrees of marked nodes have no instantiated value for this variable.

```
mark(t(L,Ds,n),Set):-
    root_same(L,Set),!,
    get_ds(Set,DsSet),
    mark_ds(Ds,DsSet).
mark(t(L,Ds,y),Set).

root_same(L,[]).
root_same(L,[t(L,_,_)|T]):-
    root_same(L,T).

mark_ds([],[]).
mark_ds([H|T],[Hs|Ts]):-
    mark(H,Hs), mark_ds(T,Ts).

get_ds([t(_,[],_)|_],[]).
get_ds(Set,[H|T]):-
    get_f(Set,Set2,H),
    get_ds(Set2,T).

get_f([],[],[]).
get_f([t(_,[H3|B],_)|T],
      [t(_,B,_)|T2],[H3|T3]):-
    get_f(T,T2,T3).
```

Changing the ambiguous parts. Summarising, the generator first generates a possible utterance. This utterance is then given as input to the monitor. The monitor calls the parser to find which parts of that utterance are ambiguous. These parts are marked in the derivation tree associated with the utterance. Finally the monitor tries to generate an utterance which uses alternative derivation trees for the marked, i.e. ambiguous, parts.

Generating an utterance given a marked derivation tree proceeds as follows. The generator simply 'repeats' the previous generation in a top-down fashion, as long as it encounters unmarked nodes. This part of the generation algorithm thus simply copies previous results. If a marked node is encountered the embedded generation algorithm is called for this partial structure. The result should be a different derivation tree than the given one. Now clearly, this may or may not be possible depending on the grammar. The next paragraph discusses what happens if it is not possible.

The following definition assumes that grammar rules are represented simply as `rule(Name, Mother, Ds)` where `Name` is the rule name, `Mother` is the mother sign and `Ds` is a list of daughter signs. The predicate `mgen` is used to generate an utterance, using a marked derivation tree as an extra guide.

```
mgen(sign(Lf,Str,S,D),t(Name,Ds,y)):-
    generate(sign(Lf,Str,S,D)),
    \+ D = t(Name,Ds,_).

mgen(sign(Lf,Str,S,D),t(Name,Ds,n)):-
    rule(Name,sign(Lf,Str,S,D),Kids),
    mgen_ds(Kids,Ds).

mgen_ds([],_).
mgen_ds([S|T],[Stree,Ttree]):-
    mgen(S,Stree),
    mgen_ds(T,Ttree).
```

Redefining locality. Often it will not be possible to generate an alternative expression by a local change as we suggested. We propose that the monitor first tries to change things as local as possible. If all possibilities are tried, the notion 'locality' is redefined by going up one level. This process repeats itself until no more alternative solutions are possible. Thus, given a marked derivation tree the monitored generation first tries to find alternatives for the marked parts of the tree. If no further possibilities exist, all markers in the trees are inherited by their mother nodes. Again the monitored generation tries to find alternatives, after which the markers are pushed upwards yet another level, etc.

The following definition of the predicate `mark_l_g(Tree, Set, Guide)` will (procedurally speaking) first construct the 'guide' `Guide` given a derivation tree `Tree` and a set of derivation trees `Set`; upon backtracking it will push the markers in the tree one level upward at the time.

```
mark_l_g(Tree,Set,Guide):-
```

```
    mark(Tree,Set),
    l_g(Tree,Guide).

l_g(Tree,Tree).
l_g(Tree,Guide):-
    one_up(Tree,Tree2),
    l_g(Tree2,Guide).

one_up(t(L,Ds,n),t(L,Ds,y)):-
    member(t(_,_,y),Ds),!.
one_up(t(L,Ds,n),t(L,Ds2,n)):-
    one_up_ds(Ds,Ds2).

one_up_ds([],[]).
one_up_ds([H|T],[H2|T2]):-
    one_up(H,H2), one_up_ds(T,T2).
```

The algorithm can be completed as follows. [2]

```
monitored_generation(LF,Sign):-
    generate(sign(LF,Str,Syn,D)),
    !,  % stick to one..
    monitor(sign(LF,Str,Syn,D),Sign).

monitor(sign(LF,Str1,Syn1,D1),
        sign(LF,Str,Syn,D)):-
    find_all_parse(Str1,TreeSet),
    (   TreeSet = [_]
    ->  Str1 = Str, Syn1 = Syn, D1 = D
    ;   mark_l_g(D1,TreeSet,Guide),
        mgen(sign(LF,Str,Syn,D),Guide),
        unambiguous(Str)
    ).

find_all_parse(Str1,TreeSet) :-
    setof(D,LF^S^parse(sign(LF,Str1,S,D),
          TreeSet).
```

Simple attachment example. In order to clarify the monitoring strategy we will now consider how an attachment ambiguity may be avoided. The following German sentence constitutes a simplified example of the sort of attachment ambiguity shown in (2).

(3) Die Männer haben heute die Frau mit dem Fernglas gesehen.
The men have today the woman with the telescope seen.
Today the men saw the woman with the telescope.

[2] In the actual implementation the predicate `find_all_parse` is complicated in order to remember which parses were already tried. If a parse has been tried before, then the predicate fails because then that result is either already shown to be ambiguous, or otherwise the corresponding solution has already been found.

Suppose indeed that the generator, as a first possibility, constructs this sentence in order to realize the (simplified) semantic representation:

$heute(mit(fernglas, sehen(pl(mann), frau))$

Let us assume that the corresponding derivation tree is the tree in figure 3. To find out whether

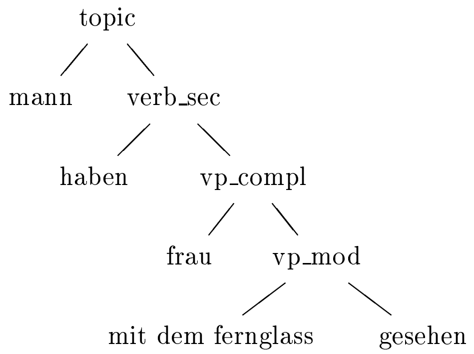

Figure 3: Derivation tree of German example

this sentence is ambiguous the parser is called. The parser will find two results, indicating that the sentence is ambiguous. For the alternative reading the derivation tree shown in figure 4 is found.

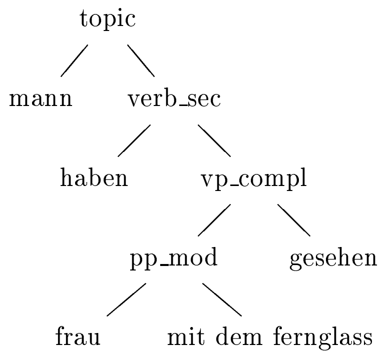

Figure 4: Derivation tree of alternative reading

The derivation tree of the result of generation is then compared with the trees assigned to the alternative readings (in this case only one), given rise to the marked derivation tree shown in figure 5.

The monitored generation will then try to find alternative possibilities at these marked nodes. However, no such alternatives exist. Therefore, the markers are pushed up one level, obtaining the derivation tree given in figure 6.

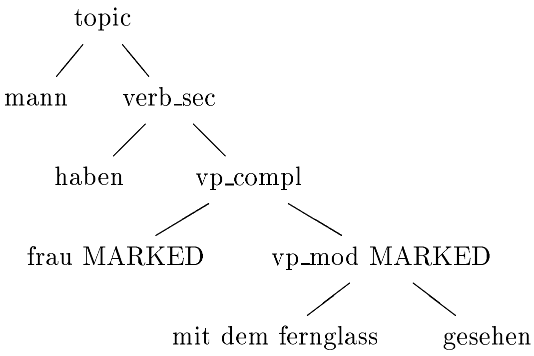

Figure 5: Marked tree of German example

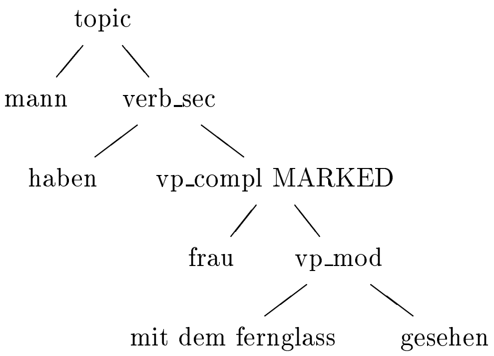

Figure 6: Markers are pushed one level upward

At this point the monitored generator again tries to find alternatives for the marked nodes, this time successfully yielding:

(4) Die Männer haben mit dem Fernglass die Frau gesehen.

At this point we may stop. However, note that if we ask for further possibilities we will eventually obtain all possible results. For example, if the markers are pushed to the root node of the derivation tree we will also obtain

(5) Mit dem Fernglass haben die Männer die Frau gesehen.

4 Discussion

Properties. Some of the important properties of our approach can be characterised as follows.

The strategy is *sound* and *complete* in the sense that no ambiguous utterances will be produced, and all un-ambiguous utterances are produced. If for a given semantic structure no un-ambiguous utterance is possible, the current strategy will not deliver a solution (it is foreseen that in such cases the planner decides what should happen).

The strategy is completely independent on the grammars that are being used (except for the reliance on derivation trees). Even more interestingly, the nature of the underlying *parsing* and *generation* strategy is not important either. The strategy can thus be used with any parsing- or generation strategy.

During the monitored generation previously generated structures are re-used, because only the ambiguous partial structures have to be regenerated.

Finally, for the proposed strategy to be meaningful, it must be the case that *reversible* grammars are being used. If this were not the case then it would not make sense to compare the derivation tree of a generation result with the derivation trees which the parser produces.

Generation of Paraphrases. In Neumann and van Noord (to appear) we discuss the implementation of a variant of the monitoring strategy, to solve the problem of the generation of paraphrases.

If parsing an utterance has lead to several readings, one way in order to determine the intended meaning is to start a clarification dialog, in which the multiple interpretations of the parsed utterance are contrasted by restating them in different form. The dialog partner is then requested to choose the appropriate paraphrase, by asking her 'Do you mean X or Y ?'.

The advantage of our approach is, that it will be ensured that the source of the ambiguity is used directly during the production of such paraphrases. Therefore, the generation of irrelevant paraphrases is avoided.

Limitations. It should be clear that monitoring involves more than the avoidance of ambiguities. Levelt (1989) discusses also monitoring on the conceptual level and monitoring with respect to social standards, lexical errors, loudness, precision and others. Obviously, our approach is restricted in the sense that no changes to the input LF are made.

Meteer (1990) makes a strict distinction between processes that can change decisions that operate on intermediate levels of representation (*optimisations*) and others that operate on produced text (*revisions*). Our strategy is an example of revision. Optimisations are useful when changes have to be done during the initial generation process. For example, in Neumann and

Finkler (1990) an incremental and parallel grammatical component is described that is able to handle under-specified input such that it detects and requests missing but necessary grammatical information.

Implementation. In Levelt (1989) and Meteer (1990) the need for revision respectively monitoring is discussed in detail although they describe no implementations. As far as we know our approach is the first implementation of revising a produced utterance in order to find an un-ambiguous alternative. The underlying parser and generator are described in Shieber *et al.* (1990) and van Noord (1991). We are using lexicalized unification-based grammars for German and Dutch.

Acknowledgements

This research work has been partially supported by the German Science Foundation in its Special Collaborative Research Programme on Artificial Intelligence and Knowledge Based Systems (SFB 314, Project N3 BiLD), and by the DFKI in the project DISCO, funded by the German Ministry for Research and Technology under Grant-No.: ITW 9002.

References

Douglas E. Appelt. Bidirectional grammars and the design of natural language generation systems. In Y. Wilks, editor, *Theoretical Issues in Natural Language Processing*, pages 206–212. Hillsdale, N.J.: Erlbaum, 1989.

K. De Smedt and G. Kempen. Incremental sentence production, self–correction and coordination. In G. Kempen, editor, *Natural Language Generation*, pages 365–376. Martinus Nijhoff, Dordrecht, 1987.

Jerry A. Fodor. *The Modularity of Mind: An Essay on Faculty Psychology.* A Bradford Book, MIT Press, Cambridge, Massachusetts, 1983.

Anthony Jameson and Wolfgang Wahlster. User modelling in anaphora generation: Ellipsis and definite description. In *ECAI*, pages 222–227, Orsay, 1982.

Aravind K. Joshi. Generation – a new frontier of natural language processing? In *Theoretical Issues in Natural Language Processing 3*, New Mexico State University, 1987.

Willem J. M. Levelt. *Speaking: From Intention to Articulation.* MIT Press, Cambridge, Massachusetts, 1989.

Marie M. Meteer. *The Generation Gap – the problem of expressibility in text planning.* PhD thesis, University of Massachusetts, 1990.

Günter Neumann and Wolfgang Finkler. A head-driven approach to incremental and parallel generation of syntactic structures. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, pages 288–293, Helsinki, 1990.

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

Günter Neumann. Reversibility and modularity in natural language generation. In *Proceedings of the ACL Workshop on Reversible Grammar in Natural Language Processing*, pages 31–39, Berkeley, 1991.

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

Gertjan van Noord. Head corner parsing for discontinuous constituency. In *29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, 1991.

R. Wilensky, Y. Arens, and D. Chin. Talking to unix in english: An overview of uc. *Communications of the ACM*, pages 574 – 593, 1984.