1

REVERSIBILITY AND SELF-MONITORING IN NATURAL LANGUAGE GENERATION

Günter Neumann*, Gertjan van Noord**

** Deutsches Forschungszentrum für Künstliche Intelligenz*
Saarbrücken
*** Vakgroep Alfa-informatica*
Rijksuniversiteit Groningen

ABSTRACT

This paper shows how the use of reversible grammars may lead to efficient and flexible natural language parsing and generation systems.

In particular a mechanism is described which ensures that only non-ambiguous utterances are produced. This mechanism uses the parsing component to monitor the generation component. The relevant communication between the two components is performed using derivation trees. For this reason the proposed mechanism only makes sense for systems in which a single grammar is used for both parsing and generation.

Furthermore we define a variant of the monitoring strategy which can be used to paraphrase a given input sentence (for interactive disambiguation). In this case, the generation component is used to guide the parsing system. Again the proposed technique is possible only in the case of a single, reversible grammar.

1 INTRODUCTION

Recently, there is a strong tendency to use the same grammar for parsing and generation. The general motivations for reversible grammars are reviewed in section 2.

Most of the generation systems use grammars that are specifically designed for generation purposes (cf. [0], [0], [0], [0]). The purpose of this paper is to show that the use of a reversible architecture for grammatical processing has important influences on the generation task. A thorough-going use of a reversible grammar within a natural language generation system affects the separation into strategic and tactical components. On the one hand, problems with this separation emerge; on the other hand reversible architectures will serve as an important (linguistic) basis to achieve first solutions to the problems.

In section 3 we discuss in more depth important problems and restrictions with the modular design of current natural language generation systems. The problem can be summarized as follows. Since a tactical component is mainly guided by the compositional structure of the semantic input, it cannot control by itself those particular combinations of partial strings of the whole utterance which will lead to alternative derivations when the hearer is parsing this utterance. This means that possible ambiguities are out of the generator's view, and will only arise during parsing.

In order to overcome these problems we propose an *integrated approach* of parsing and generation based on reversible grammars in section 4. A major point of this approach is that one mode of operation (e.g., parsing) is used for *monitoring* the other mode (e.g, generation).

In sections 5 a mechanism is presented which ensures that the generator, if so desired, only generates a non-ambiguous utterance for a given semantic specification. This mechanism uses a *monitor* which is based on the parsing component. The monitor gives feedback to the generation system in the form of annotated *derivation trees* which locate the current ambiguities. The generator uses such decorated derivation trees to avoid those ambiguities. Clearly this use of derivation trees only makes sense if the derivation trees have the same meaning for both components, i.e. they refer to the same reversible grammar.

In section 6 we describe the use of the integrated approach for a mechanims which generates paraphrases in an interactive disambiguation component. Again the parsing and generation component communicate through the use of derivation trees. Therefore the proposed mechanism is strongly dependent on the use of a single reversible grammar.

2 REVERSIBLE GRAMMARS

Constraint-based grammars have become increasingly popular within the field of natural language processing. One of the reasons for this success is that constraint-based grammars are completely *declarative*; that is, the grammar only states facts of the language it describes, without stating how such a grammar should be used for parsing or generation. Such declarative grammars, for this reason, provide for an abstract level of language description, and are easy to understand, and hence relatively easy to debug, extend, and re-use in other applications.

Because constraint-based grammars do not enforce a specific processing regime, it is possible to conceive of constraint-based grammars, which can be used *both* for parsing and generation. Such grammars may be called *reversible* [0]. Section 2.2 defines the notion *reversibility* somewhat more precisely, and discusses some of its properties. We will define a reversible grammar as a grammar for which parsing and generation are both guaranteed to terminate. The notion of a grammar that can be used both for parsing and generation is intuitively appealing; but we now provide explicit motivation for the use of reversible grammars (cf. [0]).

2.1 Motivations

Motivations for reversible grammars can be divided into linguistic, language technological, and psychological motivation. The different kinds of motivation are now discussed in turn.

Linguistic motivation

If we assume a reversible grammar, then we make two claims. The first claim is that language should be described by a single grammar (rather than a different grammar for understanding and a different grammar for production). The second claim is that this single grammar, moreover, can be used effectively both for parsing and generation.

The first claim can be motivated linguistically as follows. The primary goal of (theoretical) linguistics is to characterize languages. How such languages are used by humans (or computers) are different questions. Thus, given a language such as English, the primary goal of linguistics is to define the possible English utterances and their corresponding meanings. Thus a single language should be described by a single grammar.

The second claim, that this single grammar should moreover be (effectively) reversible, can be motivated as follows. Given that the goal of linguistics is to define the relationship between utterances and meanings, it seems that, to check a possible theory, we should be able to find out the predictions such a theory makes. That is, for a given utterance it should be possible to 'know' what the possible meanings are, according to the grammar (and vice versa). Thus, for each grammar, we want to be able to compute the corresponding meaning representations for a given utterance, and to compute the corresponding utterances for a given meaning representation.

Language technological motivations

In order to build practical NLP systems, the use of reversible grammars can be motivated, both on methodological ground (as a means to obtain 'better' systems) and practical grounds (as a means to obtain systems in a more efficient way).

Methodological

An important motivation for reversible grammars in NLP is of a methodological nature. If we are to use grammars both for parsing and generation we are *forced* to write grammars declaratively. This in turn implies that a more abstract analysis of the linguistic facts is necessary in the general case. If we are to write a declarative grammar which is used only for, say, parsing, it is quite easy to 'cheat' and 'adapt' the grammar to the parsing algorithm that is being used. In a reversible grammar this is much harder because at any moment there are two algorithms for which the grammar must be applicable.

The claim is that the use of reversible grammars will eventually lead to better grammars. For example, a grammar that is written for parsing will typically over-generate quite a lot; i.e. it will assign logical forms to sentences that are in fact ungrammatical. Not only is such a state of affairs undesirable if we are interested in describing the relation between form and meaning, it can also be argued that over-generation of parsing is a problem, even if we are only interested in parsing well-formed utterances, because over-generation typically leads to 'false ambiguities'.

An example may clarify this point (this example was actually encountered in the development of a working system). Consider a grammar for English that is intended to handle auxiliaries. Suppose that the English auxiliaries are analyzed as verbs that take an obligatory VP-complement. Moreover each auxiliary may restrict the *vform* (participle, infinite) of this complement. This allows the analysis of sentences such as

(1) Graham *will have been traveling* with his aunt

However, the possible *order* of English auxiliaries (eg. 'have' should precede 'be') is not accounted for and the analysis sketched above will for example allow sentences such as

(2) *Graham *will be having traveled* with his aunt

In the case of a reversible grammar such constructions should clearly not be generated, hence the analysis will be changed accordingly. However, even if the grammar is only used for parsing, this analysis runs into problems because it will assign two meanings to the sentence:

(3) Graham is having grilled meat

The meanings that are assigned, roughly correspond to the sentences:

(4) a. Graham ordered grilled meat
b. Graham has been grilling the meat

where only the first reading is acceptable. Thus, over-generation is not acceptable, even for grammars which are used only for parsing, because over-generation typically implies that 'false ambiguities' are produced.

In some cases, the over-generation may also lead to an explosion of local possibilities during parsing. If the grammar is more constrained, then this may sometimes be good from an efficiency point of view, because in that case there are less local ambiguities the parser has to keep track of.[1]

[1] Clearly this is not necessarily the case: a finite state grammar, which recognizes a superset of English need not be constrained at all, but can be parsed very efficiently ...

Thus, a reversible architecture may be a useful methodology to obtain a good parsing system.

Similarly, a grammar that is built for generation will usually under-generate; i.e. it will only generate a canonical sentence for a given logical form, even if there are several possibilities. Again, from a theoretical perspective this is clearly undesirable. It can be argued that a reversible architecture also leads to a better generation system. It has often been argued that, in particular situations, a generation system should produce an un-ambiguous utterance. In other situations, however, ambiguous utterances are harmless because the hearer can easily disambiguate the utterance. In this article we propose a model of language production in which a generator instructs its grammatical component whether or not it should check for ambiguity of its proposed utterance. The grammatical component, quite independently, computes an un-ambiguous utterance if so desired. For this model to be possible at all, it must be the case that the grammatical component has at its disposal several utterances for a given semantic structure in order to find in a given situation the most appropriate one. Note that 'ambiguity' might be only one of several parameters that may or may not be instantiated in a given situation. Summarizing this point, the claim is that under-generation is undesirable from the point of view of extendability.

Consistency

Consider an NLP system which is used both to convey messages to a user, and to understand the requests of the user. Necessarily, the sentences the system is able to produce and to understand are somewhat limited, given the state of the art. This may not be problematic for a user, as she might adapt herself to this restriction. However, a user will invariably assume that she can use the sort of sentences the system produces itself. That is, a reasonable constraint for such a system is that the sentences it produces is a subset of the sentences it is able to understand. In a reversible grammar the problem to check that the parser and generator are *consistent* in this respect (i.e. that the system should be able to understand those types of sentences, which it produces itself) is solved automatically; hence no special consistency checking device needs to be considered.

Practical

From a practical point of view it may be argued that it is cheaper to construct one reversible grammar than two separate grammars. The same argument can then be applied to the costs of maintaining the grammars. These two arguments of course extend to the lexical entries in the grammar: in a reversible architecture only one lexicon needs to be built and maintained, although clearly non-reversible grammars may share their lexicon too.

Furthermore, a reversible grammar provides grammar writers with a very effective debugging tool. To check whether the grammar does accept ungrammatical sentences it is possible to use the generator to see whether such ungrammatical sentences are being produced. Clearly, this technique cannot be used to ensure that a grammar does not produce ungrammatical sentences, however in practice such a tool turns out to be quite useful, as many errors in the grammar are detected this way.

Psychological motivation

An interesting question might be whether humans base their language production and language understanding on a single body of grammatical knowledge. Clearly, this would explain why humans speak the same language they understand and vice versa. Some empirical evidence for shared processors or facilities is discussed in [0], [0] and [0].

In practice speakers often understand sentences they would never produce. This observation may have several reasons.

Many differences are due to the fact that people often are able to understand otherwise mysterious utterances, because of the context and situation — using intelligence rather than grammatical knowledge. For example, hearers may understand an utterance even if the utterance contains a word they hear for the first time (and hence they could never have produced such an utterance), provided the situation or context makes it clear what this word means. Thus 'learning' often takes over from natural language understanding proper.

Alternatively, it may simply be the case that people understand sentences, they never utter, because they do not come up with the meaning in the first place. This situation might occur, either because they are not able to come up with the meaning (Einstein's case), or because they do not want to come up with that meaning (Dan Quayle's case) [2]. The first time that Einstein explained to his colleagues the relativity theory they were probably able to understand him. However none of them would have been *able* to produce Einstein's utterances. As another example, consider the case where someone uses special stylistic effects. A hearer may recognize the social register associated with these effects; this thus will be part of the 'meaning' of the utterances of that speaker. However, the hearer may belong to a different social class, and hence its language components will generally be instructed with a different 'meaning' representation to that effect. Thus, it seems that some of the differences between understanding and production are not to be explained linguistically, but are due to a difference at another level of cognitive behavior.

Thus, maybe it is possible to maintain that the grammatical part of language understanding and production can be modeled by assuming it is based on a reversible grammar. On the other hand, if it is not possible to maintain this claim in its full generality, then I believe that the model proposed here provides a good starting point for a more realistic model of language behavior.

2.2 Reversibility

Intuitively, we call a program 'reversible' if it is capable of both parsing and generation on the basis of a single characterization of the relation between semantic structures and phonological structures. The following definitions of *reversibility* are meant to be independent of the way we go about achieving a reversible natural language processing component. Furthermore, the definitions abstract away from the actual representations between which we are defining relations.

A program or system will be called r*-reversible* iff it computes a binary relation r in both directions. The idea is that, given an element of a pair in the relation, the program computes the corresponding element(s) of that pair. To encode the 'direction' of the relation I assume that the input for the program consists of a pair $\langle dir, x \rangle$ where dir represents the direction which the program should compute. The value of dir is either 0 or 1. If the value is 0, then the program computes the relation from left to right; if the value is 1 then the program computes the relation from right to left. [3]

[2] This characterization is due to Jim Barnett. It may be possible to understand the things D.Q. says, but one would never want to talk like him.

[3] Note that it usually will be quite clear from the input in which direction the relation is to be computed.

Definition 1 (Compute a relation in both directions) *A program P computes a relation r in both directions, iff P enumerates for a given input $\langle dir, e\rangle$ the set*

$$\{x| \begin{array}{l} \langle e,x\rangle \in r \wedge dir = 0 \vee \\ \langle x,e\rangle \in r \wedge dir = 1 \end{array} \}$$

Definition 2 (Reversible)

- *A program P is r-*reversible *iff P computes r in both directions.*
- *A relation r is* reversible *iff there exists an r-reversible program.*

Consider the case where r is the relation between phonological and semantic representations defined by some grammar. In this case a program is said to compute this relation in both directions (i.e. the program is reversible w.r.t. this relation) iff for a given phonological representation the program returns the corresponding semantic representation; for a given semantic representation the program returns the corresponding phonological representations. Such a program may consist of a parser and a generator (depending on the value of *dir*), or alternatively the program consists of a single uniform algorithm.

Systems in which the relation between phonological and semantic representations is defined procedurally are seldom reversible in this respect, because it is very difficult to make sure that the program indeed computes the same relation in both directions. On the other hand, a system based on a single *declarative grammar* necessarily is reversible.

Arguably, the above defined notion of reversibility is somewhat weak. According to the definition above, any recursively enumerable relation is reversible. However, in practice it is often the case that a grammar that is developed from a single perspective (eg. parsing perspective) is completely useless in the other direction because it simply fails to terminate in all interesting cases (let alone efficiency considerations). Therefore, we define what it means for a relation to be *effectively reversible*. A relation is effectively reversible if it can be effectively computed in both directions, i.e. there exists a program computing the relation in both directions, and furthermore the program always halts. In the terminology of [0], we require that there exists an *algorithm* computing the relation.

Definition 3 (Effectively reversible)

- *A program P is effectively r-reversible iff*
  - *P is r-reversible; and*
  - *P is guaranteed to terminate (for every input).*
- *A relation r is* effectively reversible *iff there exists an effectively r-reversible program.*

If we use the term *reversible* in the remainder of this article, then we will invariably mean *effectively reversible.*

3 MODULARITY IN GENERATION SYSTEMS

It is widely accepted that one may divide the problem of natural language generation (NLG) into two subtasks:

- determination of the content of an utterance
- determination of its linguistic realization

This 'divide and conquer' view of generation is the base of current architectures of systems. With few exceptions (e.g., [0]) the following two components are assumed:

- 'what to say' part (*strategic component*)
- 'how to say it' part (*tactical component*)

Currently, in systems where the separation is advocated it is assumed that the strategic component is able to provide all information needed by the tactical component to make decisions about lexical and syntactic choices [0, 0, 0, 0]. As a consequence, this implies that the input for tactical components is tailored to determine uniquely a good sentence, making the use of powerful grammatical processes redundant. In such approaches, tactical components are only front–ends while the strategic component needs detailed information about the language to use.

In [0] it is shown that the use of a reversible grammar affects the modular status of a NLG in such a way that only the tactical component should be concerned with the specific details of a grammar and the strategic component should only perform general reasoning and planning tasks. In this view, the tactical component has the same status for the production process as the parser has for the understanding process.

Consequently, the strategic component cannot completely control the tactical component. For example, the following can happen. A message which is constructed precisely enough to satisfy the strategic component's goal can be under-specified from the tactical viewpoint. In particular, the generator can run into the risk of being misunderstood because of the produced utterance's ambiguity.

For example, if the strategic component specifies the following structure SEM as input to the tactical component:[4]

$$\left[\ sem : \left[\begin{array}{ll} cont : & \left[\begin{array}{l} reln : remove' \\ agent : you' \\ patient : folder' \\ instrument : system\_tools' \end{array}\right] \\ conx : & \left[\ speech\_act : imperative\ \right] \end{array}\right]\ \right]$$

then a possible utterance is 'Remove the folder with the system tools' with the corresponding derived grammatical structure where the PP 'with the system tools' is an adjunct to the VP:

$$\left[\begin{array}{ll} \multicolumn{2}{l}{phon : \langle \textit{remove the folder with the system tools} \rangle} \\ \multicolumn{2}{l}{synsem : S[imp]} \\ dtrs : & \left[\begin{array}{ll} head : & \left[\begin{array}{l} phon : \langle \textit{remove} \rangle \\ synsem : VP[fin] \end{array}\right] \\ comp : & \langle \left[\begin{array}{l} phon : \langle \textit{the folder} \rangle \\ synsem : NP[acc] \end{array}\right] \rangle \\ adjunct : & \langle \textit{with the system tools} \rangle \end{array}\right] \end{array}\right]$$

[4]We are using an HPSG-like notation close to that of [0].

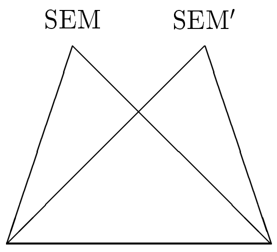
From the generator point of view this utterance is grammatical reflects exactly what the generator wants to express. For the hearer there also exists however the alternative grammatical structure where the PP 'with the system tools' is a nominal adjunct:

$$\begin{bmatrix} phon : \langle \textit{remove the folder with the system tools} \rangle \\ synsem : S[imp] \\ dtrs : \begin{bmatrix} head : \begin{bmatrix} phon : \langle \textit{remove} \rangle \\ synsem : VP[fin] \end{bmatrix} \\ comp : \left\langle \begin{bmatrix} phon : \langle \textit{the folder with the system tools} \rangle \\ synsem : PP \\ dtrs : \begin{bmatrix} head : \langle \textit{with the system tools} \rangle \\ comp : \langle \textit{the folder} \rangle \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

with the semantic reading SEM′:[5]

$$\begin{bmatrix} sem : \begin{bmatrix} cont : \begin{bmatrix} reln : remove' \\ agent : you' \\ patient : \begin{bmatrix} index : x \\ restr : folder'(x) \wedge \\ \qquad with'(x, system\_tools') \end{bmatrix} \end{bmatrix} \\ conx : \begin{bmatrix} speech\_act : imperative \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The whole situation can graphically be represented as follows:



Remove the folder with the system tools

[5]If the generator is part of an intelligent help system, the choice of this reading could have tremendous effects on the system itself.

The left triangle represents the domain of the derivation between the semantic structure SEM and the utterance ‘Remove the folder with the system tools’ obtained during generation. Both triangles represent the domain of derivation between the utterance and the semantic structures SEM and SEM′ computed during parsing.

Now the problem can be stated as follows. Since a tactical component is mainly guided by the compositional structure of the semantic input, it cannot control by itself those particular combinations of partial strings of the whole utterance which will lead to alternative derivations when the hearer is parsing this utterance. This means that possible ambiguities are out of the generator’s view, and will only arise during parsing.

Of course, one could argue that if the generator had produced the utterance ‘Remove the folder by means of the system tools’ instead of ‘Remove the folder with the system tools’ then the kind of ambiguity exemplified above would not occur. Choosing the former instead of the latter in order to avoid ambiguity would mean that the strategic component is able to foresee that the generation process will run into the risk of generating an ambiguity, and hence of conveying misinformation. The problem here depends on the alternative possible realizations of the instrument role, namely ‘with’ or ‘by means of’. The strategic component could have chosen ‘by means of’ for some reasons internal to it (e.g., stylistic reasons, preferences, etc.) but not because it could foresee the ambiguity of ‘with’. In other words, given the modular design, the fact that at some point a potentially ambiguous LF surfaces as a un-ambiguous string cannot be assumed to be due to the fact that the ambiguity was foreseen, just other factors, independent from that, made the utterance unambiguous. If the strategic component chose ‘by means of’ in order to restrict the set of possible derivations during parsing, this would mean that it is able to make decisions because of grammatical reasons.

The particular realization of the instrument role is not always relevant in order to avoid ambiguity. For example, in German (a language with relatively free word order) it would also be possible to utter:

‘Mit den Systemwerkzeugen den Ordner löschen’
‘[With the system tools] [the folder] remove’
(which can only mean *Remove the folder by means of the system tools*)

In this case the utterance is disambiguated by means of a specific ordering of the constituents. But now the same problem holds: Without detailed grammatical background the strategic component would not be able to specify the correct ordering in order to avoid ambiguity.

Summarising, it should be clear now that the strategic component cannot have this kind of control because otherwise this would blur the modular design of a generation system mentioned above. Fortunately, in many situations of communication a speaker need not 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 that she would otherwise ask for clarification (Nevertheless, in the next section we show that the same problem mentioned above occurs also during clarification dialogs). However, an adequate generation system should also be able to avoid the generation of ambiguous utterances in some specific situations, e.g., when utterances refer to actions that have to be performed directly or in some specific dialog situations. As long as the strategic component has no detailed knowledge of a specific grammar it could not express 'choose this particular form to avoid ambiguity'. Therefore it can happen that the intended message will not be conveyed.

4 INTEGRATION OF PARSING AND GENERATION

A promising approach to overcome these problems is *to integrate generation and parsing* in a strict way. By this we mean:

- the use of resulting structures of one direction directly in the other direction,
- the use of one mode of operation (e.g., parsing) for monitoring the other mode (e.g., generation).

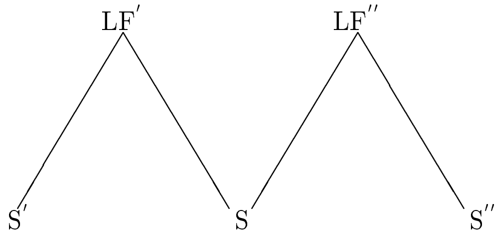
A main thesis of this paper is that the best way to achieve such integrated approach is *to use the same grammar* as the linguistic basis for both processes.

Monitoring

In order to maintain a modular design additional mechanisms are necessary to perform some *monitoring* of the generator's output. The need for such mechanisms during natural language generation has already been noted by several authors [0, 0, 0]. For example, [0] points 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 Levelt's model parsing and generation are performed in an isolated way by means of two different grammars. In such flow of control the complete structure has to be generated again if ambiguities are detected (by parsing the utterance) that have to be avoided. The problem with this view is that generation of unambiguous utterances can be very inefficient, because the relevant sources of the ambiguous utterance are not used to guide the generation process.

The same can happen during the generation of paraphrases. If for example an intelligent help-system that supports a user by using an operation system (e.g. Unix, [0]), receives as input the utterance 'Remove the folder with the system tools' then the system is not able to perform the corresponding action directly because it is ambiguous. But the system could start a clarification dialog and ask the user 'Do you mean "Remove the folder by means of the system tools" or "Remove the folder that contains the system tools"?'. This situation is summarised in figure 1 (LF$'$ and LF$''$ symbolise two readings of S).



S:	Remove the folder with the system tools
S$'$:	Remove the folder by means of the system tools
S$''$:	Remove the folder that contains the system tools

Figure 1 Relationship between ambiguities and paraphrases

As long as parsing and generation are performed in an isolated way the generator is not able to consider the source of the ambiguity such that it is ensured that only relevant paraphrases will be produced. For example, the generator could produce a paraphrase like 'Do you mean: "Delete the folder with the system tools"' which does not help to clarify the situation.

Self-Monitoring and Reversible Grammars

In order to be able to take into account relevant sources of ambiguities the basic idea of the integrated approach introduced in this paper is to operate with *derivation trees* obtained during the generation and parsing step by means of the same grammar.

The basic step of the approach is as follows: The monitor compares in each step the resulting structures of the generation process with the corresponding structures from parsing maintained in the alternative derivation trees in a top-down way (We will now assume that two derivation trees P_1 and P_2 are obtained during parsing). Suppose that LF' is specified as the input to the generator. In the case where the generator encounters alternative grammatical structures to be expanded, the monitor guides the generator by means of inspection of the corresponding derivation trees. In the case where actual considered parts p_1 and p_2 of P_1 and P_2 (e.g., same NPs) are equal, then the generator has to choose the same grammatical structure that was used to build p_1 and p_2 (or more efficiently the generator can use the partial structure directly as a kind of compiled knowledge). In the case where a partial structure of e.g., parse tree P_1 has no correspondence in P_2 a relevant ambiguity source is detected. In this case an alternative grammatical structure has to be chosen.

At this point it should be clear that the only way in order to be able to generate 'along parsed structures' is to use a reversible grammar in both directions because grammatical structures obtained during parsing are used directly to guide the generation process.

In the next two sections we describe in detail how this mechanism is used during generation of unambiguous utterances and generation of paraphrases.

5 GENERATION OF UNAMBIGUOUS UTTERANCES

A fundamental assumption is that it is often possible to obtain an unambiguous 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 unambiguous 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' ('riverside' vs. 'money institution') a generator may produce, as a first possibility, the following sentence in the case of the first reading of 'bank'.

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

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

(6) 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 '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 for a given string or logical form. For example, the derivation trees for the two readings of 'John is standing near the bank' may look as in figure 2. 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`. For simplicity

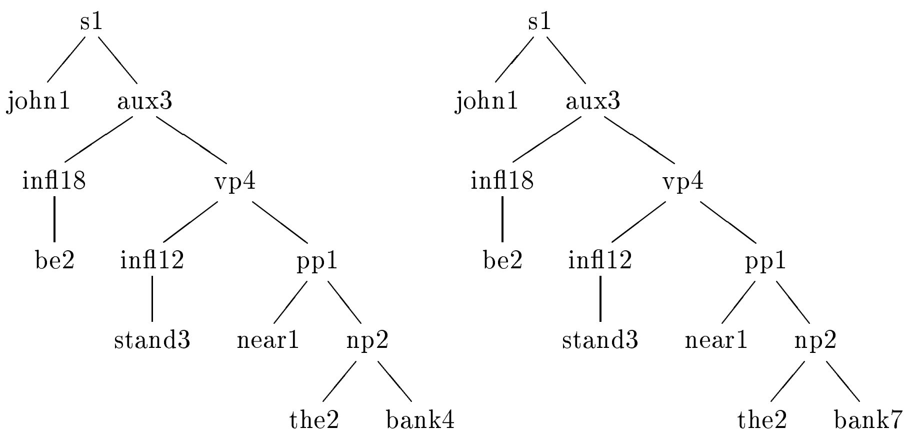


Figure 2 Derivation trees

we assume that in our examples each sign `sign(LF,Str,Syn,D)` is specified for its corresponding derivation tree `D`, where the other arguments represent the semantic information (`LF`), the string (`Str`) and syntactic information (`Syn`). 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 can now 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 can 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 (6) 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 3.

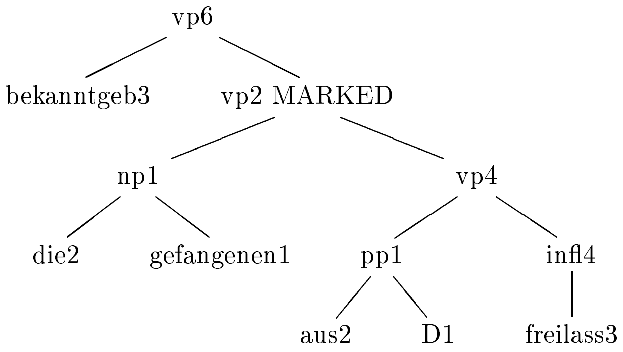


Figure 3 Marked derivation tree

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)` representing 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

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 from 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)),
    not 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 locally 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.

It is possible that by successively moving markers upwards in a marked derivation tree the root node of the tree will be marked. If also in that case no unambiguous alternative will be possible then this means that the generator is not able to compute a grammatical unambiguous alternative. In this case the whole monitored generation process terminates and the strategic component has to decide whether to utter the ambiguous structure or to provide an alternative logical form.

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).
```

Now, the whole algorithm can be completed as follows. [6]

```
monitored_generation(LF,Sign):-
    generate(sign(LF,Str,Syn,D)),
    !,  % stick to one..
    find_all_parse(sign(_,Str,_,_),_,TreeSet),
    (  TreeSet = [ ]
    -> Sign = sign(LF,Str,Syn,D)
    ;  revision(sign(LF,Str,Syn,D),TreeSet,Sign)
    ).

revision(sign(LF,Str1,Syn1,D1),TreeSet,sign(LF,Str,Syn,D)):-
    mark_l_g(D1,TreeSet,Guide),
    mgen(sign(LF,Str,Syn,D),Guide),
    unambiguous(sign(_,Str,_,_)).

find_all_parse(Sign,SignSet,TreeSet) :-
    setof(Sign,parse(Sign),SignSet),
    extract_trees(SignSet,TreeSet).

unambiguous(Sign):-
    find_all_parse(Sign,[One],_).
```

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

[6] In the actual implementation the predicate `find_all_parse` is complicated in order to remember which parses where 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.

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, eventually pushing the markers successively upwards.

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 (6).

(7) Die Männer haben die Frau mit dem Fernglas gesehen.
The men have the woman with the telescope seen.

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

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

The corresponding derivation tree is the left tree in figure 4. To find out whether 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 right derivation tree shown in figure 4 is found. 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.

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

(8) Die Männer haben mit dem Fernglas 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

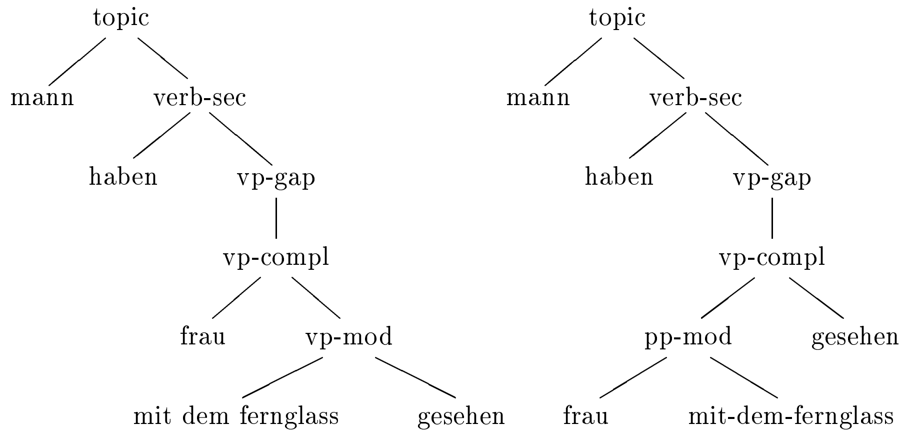


Figure 4 Derivation trees of the simple attachment example

(9) Mit dem Fernglas haben die Männer die Frau gesehen.

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 re-generated.

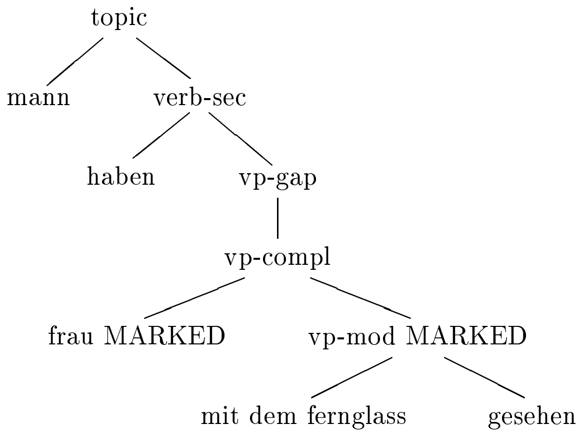


Figure 5 Marked tree of German example

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.

6 GENERATION OF PARAPHRASES

When parsing of an utterance yields several readings, one way in order to determine the intended meaning is to start a clarification dialog. During such a special dialog situation the multiple interpretations of the parsed utterance are contrasted by restating them in different text forms. Now, the dialog partner who produced the ambiguous utterance is requested to choose the appropriate paraphrase, e.g., by asking her ‘Do you mean X or Y ?’.

This situation has already been exemplified in section 4 fig. 1. In this example, parsing of S (‘Remove the folder with the system tools’) has lead to two readings LF$'$ and LF$''$. The multiple semantic forms are then paraphrased by means of the utterances S$'$ and S$''$ (‘Do you mean “Remove the folder by means of the

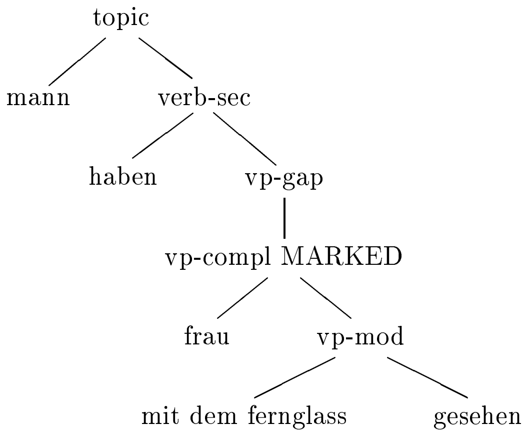


Figure 6 Markers are pushed one level upward

systems tools" or "Remove the folder that contains the system tools"?').

A naive version

A first naive algorithm that performs generation of paraphrases using a reversible grammar can be described as follows. Consider the situation in fig. 1. Suppose S is the input for the parser then the set

$$\{(\mathrm{S}, \mathrm{LF}'), (\mathrm{S}, \mathrm{LF}'')\}$$

is computed. Now LF$'$ and LF$''$ are respectively given as input to the generator to compute possible paraphrases. The sets

$$\{(\mathrm{LF}', \mathrm{S}'), (\mathrm{LF}', \mathrm{S})\}$$

and

$$\{(\mathrm{LF}'', \mathrm{S}), (\mathrm{LF}'', \mathrm{S}'')\}$$

result. By means of comparison of the elements of the sets obtained during generation with the set obtained during parsing one can easily determine the two paraphrases S' and S'' because of the relationship between strings and logical forms defined by the grammar. Note that if this relationship is effectively reversible (see section 2) then this problem is effectively computable.

This 'generate-and-test' approach is naive because of the following reasons. Firstly, it assumes that all possible paraphrases are generated at once. Although 'all–parses' algorithms are widely used during parsing in natural language systems a corresponding 'all–paraphrases' strategy is not practical because in general the search space during generation is much larger (which is a consequence of the modular design discussed in section 3). Secondly, the algorithm only guarantees that an ambiguous utterance is restated differently. It is possible that irrelevant paraphrases are produced because the source of the ambiguity is not used directly.

A suitable strategy

The crucial point during the process of generation of paraphrases is that one has not only to guarantee that an ambiguous utterance is restated differently but also that only *relevant* paraphrases are to be produced that appropriately resolve structural ambiguities.

In order to be able to take into account the source of ambiguity obtained during parsing the basic idea of the proposed approach is to generate paraphrases along 'parsed' structures. Suppose that parsing of an utterance has yielded two interpretations LF' and LF'' with corresponding derivations trees d_1 and d_2. It is now possible to generate a new utterance for each logical form LF_i by means of the monitored generation algorithm described in the previous section. In this case, the corresponding derivation tree d_i of LF_i is marked by means of the others. The so marked tree is then used to 'guide' the generation step as already known.

The paraphrasing algorithm in detail

Because most of the predicates to use are already defined in section 5 as well as the definitions of signs and rules we can directly specify the top-level predicate `interactive_parsing` as follows:

```
interactive_parsing(Str, Sign):-
    find_all_parse(sign(_,Str,_,_), SignSet, TreeSet),
    (  SignSet = [Sign]
    -> true
    ;  generate_paraphrases(SignSet, TreeSet, Paraphrases),
       ask_best_answer(SignSet, Paraphrases, Sign)
    ).
```

The predicate `find_all_parse` computes all possible parses of a given string `Str`, where `TreeSet` are all corresponding derivation trees extracted from the set of the parsed structures `SignSet`. If the parser obtains multiple interpretations then for each element of `SignSet` a paraphrase has to be generated. This is done by means of the predicate `generate_paraphrases`, whose definition will be given below. All computed `Paraphrases` are then given to the user who has to choose the appropriate paraphrase. The corresponding logical form of the chosen `Sign` determines the result of the paraphrasing process.

For each parsed sign of the form `sign(LF,Str,Syn,D)` a paraphrase is generated in the following way: First its derivation tree `D` is marked by means of the set of derivations trees contained in `TreeSet`. The resulting marked derivation tree `Guide` is then used in order to guide the generation of the sign's logical form `LF` using the predicate `mgen`. Note, that this directly reflects the definition of the predicate `revision`, which definition was given in the previous section. Therefore we can simply specify the definition of the predicate `generate_paraphrases` as follows:

```
generate_paraphrases([], _, []).

generate_paraphrases([Sign|ParsedSigns], TreeSet, [Paraphrased|T]):-
    revision(Sign,TreeSet,Paraphrased),
    !,  % one alternative for each reading
    generate_paraphrases(ParsedSigns, TreeSet, T).
```

A simple example

In order to clarify how the strategy works we consider the attachment example of section 5 again. Suppose that for the sentence

(10) Die Männer haben die Frau mit dem Fernglas gesehen.

The men has the woman with the telescope seen.

the parser has determined the derivation trees in figure 4 with corresponding (simplified) semantic representations:

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

for the left and

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

for the right tree. For the first reading the paraphrase

(11) Die Männer haben mit dem Fernglas die Frau gesehen.

is generated in the same way described in section 5. In this case the left tree of figure 4 is marked by means of the right one.

In order to yield a paraphrase for the second reading, the right derivation tree of figure 4 is marked by means of the left one. In this case markers are placed in the right tree at the nodes named 'pp_mod' and 'gesehen'. If the grammar allows to realize 'mit(frau, fernglas)' using a relative clause then the paraphrase

(12) Die Männer haben die Frau, die das Fernglas hat, gesehen.
The men have the woman, who the telescope has, seen.

is generated. Otherwise, the markers are pushed up successively to the root node 'topic' of that tree yielding the paraphrase:

(13) Die Frau mit dem Fernglas haben die Männer gesehen.
The woman with the telescope have the men seen.

Now, the produced paraphrases are given to the user who has to choose the appropriate one. In the current implementation this is simply done by entering the corresponding number of the selected paraphrase.

Properties

In principle the same properties as those already discussed for the monitored generator are valid. This means, that only unambiguous paraphrases are generated. Therefore it is guaranteed that the same paraphrase is not produced for different interpretations. This is important because it could be the case that a paraphrase, say $S^{'}$ is also ambiguous such that it has the same interpretations as S. Therefore it could happen that the same utterance $S^{'}$ is generated as a paraphrase for both $LF^{'}$ and $LF^{''}$. For example in German the following sentence:

(14) Den Studenten hat der Professor benotet, der das Programm entwickelte.
The-ACC student-ACC has the professor marked, who developed the program.

is ambiguous because it is not clear who developed the program. If a paraphrase is to be generated, which expresses that the student developed the program, then this can be done by means of the utterance:

(15) Der Professor hat den Studenten benotet, der das Programm entwickelte.
The professor has the-ASC student-ACC marked, who developed the program.

But this utterance has still the same ambiguity. This means, that one has to check also the ambiguity of the paraphrase. An unambiguous solution for the example is, e.g., the utterance:

(16) Den Studenten, der das Programm entwickelte hat der Professors benotet.
The-ACC student-ACC, who developed the program has the professor marked.

The advantage of our approach is that only one paraphrase for each interpretation is produced and that the source of the ambiguity is used directly. Therefore, the generation of irrelevant paraphrases is avoided.

Furthermore, we do not need special predefined 'ambiguity specialists', as proposed by [0], but rather use the parser to detect possible ambiguities. Hence our approach is much more independent of the underlying grammar.

7 DISCUSSION

Limitations

It should be clear that monitoring and revision involves more than the avoidance of ambiguities. [0] 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. If no alternative string can be generated then the planner has to decide whether to utter the ambiguous structure or to provide an alternative logical form.

During the process of generation of paraphrases it can happen that for some interpretations no unambiguous paraphrases can be produced. Of course, it is possible to provide the user only with the produced paraphrases. This is reasonable in the case that she can find a good candidate. But if she says e.g., 'none of these' then the paraphrasing algorithm is of no help in this particular situation.

In [0] a strict distinction is made 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 [0, 0] 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.

Comparison and Implementations

In [0] strategies for paraphrasing are described. They propose an approach where during the repeated parse of an ambiguous utterance potential sources of ambiguity can be detected. For example when in the case of lexical ambiguity a noun can be associated with two semantic classes a so called 'lexical ambiguity specialist' records the noun as the ambiguity source and the two different classes. These two classes are then explicitly used in the generator input and are realized, e.g., as modifiers for the ambiguous noun.

The only common knowledge source for the paraphraser is a high-order intensional logic language called World Model Language. It serves as the interface between parser and generator. The problem with this approach is that parsing and generation are performed in an isolated way using two different grammars. If an ambiguous utterance S needs to be paraphrased, S has to be parsed again. During this repeated parse *all* potential ambiguities have to be recognised and recorded by means of different 'ambiguity specialists'. The problem here is that also local ambiguities have to be considered that are not relevant for the whole structure.

Furthermore, the general status of their work is not clear. For example, during generation they use a procedural grammar where it is assumed that all relevant linguistic information is specified in the input of the tactical component. Our work is much more general because it is independent of the grammar and the underlying parser and generator.

In [0] and [0] the need for monitoring or revision is discussed in detail although they describe no implementations. As far as we know our approach is the first implementation that solves the problem of revising a produced utterance in order to find an unambiguous alternative. Our strategies are implemented in Prolog. The underlying parser and generator are described in [0] and [0]. We are using lexicalized unification-based grammars for German and Dutch.

8 FUTURE WORK

It is important to investigate the implications contextual information has for disambiguation of single utterances. Clearly, in many situations of communication it is not necessary to avoid an ambiguous utterance because the context forces the intended meaning. What is really needed is a generalisation of our method which takes context into account, i.e. an *contextually sensitive* method. Furthermore, such a method may also be useful for single utterances to achieve a more efficient and realistic monitoring strategy, where generation and parsing are integrated in an *incremental way.*

The need for contextual sensitivity within single utterance

The basic strategy used for generating unambiguous utterances and paraphrases described so far can be denoted as a *global method* because it operates over fully determined derivation trees of sentences. A fundamental assumption is that it is often possible to change an ambiguous utterance *locally* to obtain an unambiguous utterance with the same meaning. If we were to base an *incremental method* on this local view, where during generation already produced partial strings are monitored before the whole string is produced then we would run into problems.

Such a strategy works for an example like:

(17) Removing the folder with the system tools can be very dangerous.

Here, the relevant ambiguity of the whole utterance is forced by the partial string 'Removing the folder with the system tools'. This ambiguity can be solved by restating the partial string, e.g., as 'Removing the folder by means of the system tools' independently from the rest of the string.

However, consider the ambiguous string 'visiting relatives' which can mean 'relatives who are visiting someone' or 'someone is visiting relatives'. If this string is part of the utterance

(18) Visiting relatives can be boring.

then a local disambiguation of 'visiting relatives' is helpful in order to express the meaning of the whole utterance clearly. But if this string is part of the utterance

(19) Visiting relatives are boring.

then it is not necessary to disambiguate 'visiting relatives' because the specific form of the auxiliary forces the first reading 'relatives who are visiting someone'.

This phenomena is not only restricted on the phrasal level but occurs also on lexical level. For example, 'ball' has at least two meanings, namely 'social assembly for dancing' and 'sphere used in games'. If this word occurs in the utterance

(20) During the ball I danced with a lot of people.

then the preposition 'during' forces the first meaning of 'ball'. Therefore it is not necessary to disambiguate 'ball' locally. But, for the utterance

(21) I know of no better ball.

'ball' cannot be disambiguated by means of grammatical relations of the utterance.

The problem is that one has to control the monitor already during incremental processing of single utterances in order to decide when disambiguation of ambiguous partial structures has to take place. Technically, it is possible to check and revise the partial results of each recursive call of the generator. But, without any control, the monitor would try to disambiguate each local ambiguity; it is hard to imagine that the resulting generator would produce anything at all.

Outline of an incremental method

An utterance can only be said to be (un)ambiguous with respect to a certain *context*. The assumption is that usually an utterance which is not ambiguous w.r.t. its context will remain unambiguous if it is part of a larger utterance.

Assume we have a predicate `parse_wrt_context(Str,Sign,Cont)`, which parses a string `Str` as a sign `Sign` with respect to context `Cont`. The monitoring strategy can be revised in order to use this predicate instead of the `parse` predicate. Only those sources of ambiguities are taken into account that lead to *relevant* ambiguities w.r.t. the context.

More speculatively, it may be possible to restrict the context during the production of a partial utterance to grammatical properties, e.g. to the information associated with the *head* which selects the phrase dominating this partial utterance. Such an approach can be integrated in head-driven generators of the type described in [0].

For example, assume that for each recursive call to the generator the revised monitor is called, with an extra argument `Head` which represents the context for the `parse_wrt_context` predicate. Thus, suppose we are to generate from the logical form

$$during'(ball')$$

A head-driven generator first produces the word `during` as the head. Next a NP with logical form $ball'$ has to be generated. For this logical form the generator chooses the word `ball` which is however ambiguous. For this partial utterance the monitor is called, using the head information of 'during'. However, being an argument of the head 'during', only one of the readings of 'ball' is possible. Therefore, the monitor simply 'confirms' the choice of the generator. Thus, the assumption here is that this ambiguity will be disambiguated later on by combining this string with its head. Clearly, this need not *always* be the case — therefore this strategy can be seen as a *preference* over the search space for the generator, using a sort of structural 'look-a-head'.

Conclusion

A main objective of this paper was to show that problems with the modular design of current generation systems emerge when a reversible grammar is used. In order to maintain the modular design we have proposed an approach that is based on a strong integration of parsing and generation of grammatical structures using a reversible grammar and monitoring mechanisms. By means of such an integrated approach performing generation of unambiguous utterances as well as generation of paraphrases can be realized.

Acknowledgements

This research work has been supported by the German Science Foundation in its Special Collaborative Research Programme on Artificial Intelligence and Knowledge Based Systems (SFB 314, Project N3 BiLD). Many thanks go to Sergio Balari, John Nerbonne and Hans Uszkoreit for their valuable comments on earlier versions.

REFERENCES

[1] Douglas E. Appelt. *Planning English Sentences.* Cambridge University Press, Cambridge, 1985.

[2] 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.

[3] Stefan Busemann. *Generierung natürlicher Sprache mit Generalisierten Phrasenstruktur–Grammatiken.* PhD thesis, University of Saarland (Saarbrücken), 1990.

[4] Robert Dale. Generating receipes: An overview of epicure. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, pages 229–255. Academic Press, London, 1990.

[5] 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.

[6] Wolfgang Finkler and Günter Neumann. Popel-How: A distributed parallel model for incremental natural language production with feedback. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1518–1523, Detroit, 1989.

[7] Lyn Frazier. Shared components of production and perception. In M. A. Arbib et al., editor, *Neural Models of Language Processes*, pages 225–236. Academic Press, New York, 1982.

[8] Merrill F. Garrett. Remarks on the relation between language production and language comprehension systems. In M. A. Arbib et al., editor, *Neural Models of Language Processes*, pages 209–224. Academic Press, New York, 1982.

[9] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[10] Helmut Horacek. The architecture of a generation component in a complete natural language dialogue system. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, pages 193 – 227. Academic Press, London, 1990.

[11] Ray Jackendoff. *Consciousness and the Computational Mind.* MIT Press, Cambridge, Massachusetts, 1987.

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

[13] Martin Kay. Syntactic processing and functional sentence perspective. In *Theoretical Issues in Natural Language Processing — supplement to the Proceedings*, pages 12–15, Cambridge Massachusetts, 1975.

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

[15] David D. McDonald. Natural language generation as a computational problem: An introduction. In M. Brady and C. Berwick, editors, *Computational Models of Discourse.* MIT Press, Cambridge, Massachusetts, 1983.

[16] Kathleen R. McKeown, Michael Elhadad, Yumiko Fukomoto, Jong Lim, Christine Lombardi, Jacques Robin, and Frank Smadja. Natural language generation in comet. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, pages 103 – 139. Academic Press, London, 1990.

[17] Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press, Cambridge, 1985.

[18] Marie M. Meteer and Varda Shaked. Strategies for effective paraphrasing. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

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

[20] 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.

[21] Carl Pollard and Ivan Sag. *Head-driven Phrase Structure Grammar.* Center for the Study of Language and Information Stanford, 1993. in press.

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

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

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

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