Acknowledgement

This research begun while I was working at the department of Computational Linguistics at the University of Saarland and the project BiLD, which is supported by the German Science Foundation in its Special Collaborative Research Program on Artificial Intelligence and Knowledge Based Systems SFB 314. It continued at the German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, DFKI) in the Computational Linguistics area. I am grateful to these institutes for their support.

Many, many people have helped me not to get lost during the development of this thesis. Hans Uszkoreit, my main supervisor, provided a motivating, enthusiastic, and critical atmosphere during the many discussions we had. It was a great pleasure to me to conduct this thesis under his supervision. I also acknowledge Wolfgang Wahlster who as my second supervisor provided constructive comments during my thesis time as well as on the preliminary version of this thesis.

During many discussions and computer demos of my thesis stuff I received a lot of valuable comments from at least the following people: Jan Alexandersson, Rolf Backofen, Sergio Balari, Jim Barnett, Stephan Busemann, Gregor Erbach, Wolfgang Finkler, Dan Flickinger, Karin Harbusch, Elizabeth Hinkelman, Aravind Joshi, Walter Kasper, Martin Kay, Bernd Kiefer, Hans-Ulrich Krieger, John Nerbonne, Klaus Netter, Gertjan van Noord, Stephan Oepen, Thomas Pechmann, Norbert Reithinger, Vijay Shankar, Mark Steedman, Harald Trost, Mats Wirén.

The following people have read draft versions of the thesis: Rolf Backofen, Elizabeth Hinkelman, Bernd Kiefer, Karin Harbusch, Gregor Erbach, and Mats Wirén.

Needless to say, that I am grateful to all of my colleagues at the Computational Linguistics department and at the DFKI for their support (and tolerance); especially I am indebted to the 'DISCO people'.

Specials thanks to Gertjan van Noord for the great time during our common 'BiLD time'.

Mein innigster Dank gilt meiner Familie Eva, Kevin und Dennis. Ohne ihre Hilfe, Toleranz und ständige Aufmunterung wäre diese Arbeit nie beendet worden.

Postscriptum

This thesis has been submitted to the thesis committee in November 1994. In the meanwhile, an improved and extended version of the thesis content can be found in *G. Neumann: Interleaving Natural Language Parsing and Generation Through Uniform Processing, Journal of Artifical Intelligence 99, 1998, 121–163.*

A recent comparision of several uniform architectures (including the one proposed in this thesis) can be found in: S. Varges: Parsing und Generierung in uniformen Architekturen, Uni. Düsseldorf, 1997 (see also http://web.phil-fak.uni-duesseldorf.de/~varges/master.html). He also presents a Prolog implementation of the uniform tabular algorithm developed in this thesis.

Contents

List of Figures

Chapter 1

Introduction

In the area of natural language processing in recent years, there has been a strong tendency towards *reversible natural language grammars*, i.e., the use of one and the same grammar for grammatical analysis (parsing) and grammatical synthesis (generation) in a natural language system.

The idea of representing grammatical knowledge only once and of using it for performing both tasks seems to be quite plausible, and there are many arguments based on practical and psychological considerations for adopting such a view (in section 2.1 we discuss the most important arguments in more detail).

Nevertheless, in almost all large natural language systems in which parsing and generation are considered in similar depth, different algorithms are used – even when the same grammar is used. At present, the first attempts are being made at *uniform* architectures which are based on the paradigm of natural language processing as deduction (they are described and discussed in section 2.3 in detail). Here, grammatical processing is performed by means of the same underlying deduction mechanism, which can be parameterized for the specific tasks at hand.

Natural language processing based on a uniform deduction process has a formal elegance and results in more compact systems. There is one further advantage that is of both theoretical and practical relevance: a uniform architecture offers the possibility of viewing parsing and generation as strongly interleaved tasks. *Interleaving parsing and generation* is important if we assume that natural language understanding and production are not performed in an isolated way but rather can work together to obtain a flexible use of language. In particular this means a.) the use of one mode of operation for monitoring the other and b.) the use of structures resulting from one direction directly in the other. For example, during generation integrated parsing can be used to monitor the generation process and to cause some kind of revision, e.g., *to reduce the risk of misunderstandings*. Research on monitoring and revision strategies is a very active area in cognitive science; however, currently there exists no algorithmic model of such a behaviour. A uniform architecture can be an important step in that direction.

Unfortunately, the currently proposed uniform architectures are very inefficient and it is yet unclear how an efficiency–oriented uniform model could be achieved. An obvious problem is that in each direction different input structures are involved – a string for parsing and a semantic expression for generation – which causes a different traversal of the search space defined by the grammar. Even if this problem were solved, it is not that obvious how a uniform model could re-use partial results computed in one direction efficiently in the other direction for obtaining a practical interleaved approach to parsing and generation.

1.1 The Goals of the Thesis

The major goal of this thesis is the design and theoretical and practical investigation of a uniform computational model as the basis of efficient interleaving of natural language parsing and generation. In particular we are concerned with the following questions:

- Is it possible to define a uniform algorithm that can efficiently perform natural language parsing and generation using one and the same grammar?
- How can this algorithm be extended so that it can share partial results in both directions in order to support interleaving of parsing and generation?
- How is interleaving of parsing and generation used for performing monitoring and revision during natural language processing?

We will answer these questions as follows:

A novel uniform tabular algorithm is presented that can be used for efficient parsing and generation of constraint-based grammars without the need for compilation. The most important properties of the algorithm are:

- **Earley deduction.** The control logic of the new algorithm is based on Earley deduction, i.e., it realizes a mixed top-down/bottom-up behaviour. The new algorithm is the first one that is able to apply this strategy for parsing and generation in a real symmetric and balanced way and consequently will terminate on a larger class of reversible grammars.
- **Dynamic selection function.** The uniform algorithm uses a dynamic selection function to determine the element to process next on the basis of the current portion of the input – a string for parsing and a semantic expression for generation. This enables us, for example, to obtain a left-to-right control regime in the case of parsing and a semantic functor driven regime in the

case of generation when processing the same grammar by means of the same underlying algorithm.

- **Uniform indexing technique.** The same basic mechanism is used for parsing and generation, but parameterized with respect to the information used for indexing partial results. The kind of index causes completed information to be placed in the different state sets. Using this mechanism we can benefit from a table-driven view of generation, similar to that of parsing. For example, using a semantics-oriented indexing mechanism during generation massive redundancies are avoided, because once a phrase is generated, we are able to use it in any position within a sentence.

- **Item sharing.** We present a new method of grammatical processing which we term *item sharing*. The basic idea is that items computed in one direction are automatically made available for the other direction as well. The item sharing approach is based on the uniform indexing technique mentioned above and is realized as a straightforward extension of the uniform tabular algorithm. The relevance of this novel method is demonstrated when the new performance model is presented.

Since the only relevant parameter our uniform tabular algorithm has with respect to parsing and generation is the difference in input structures the basic differences between parsing and generation are simply the different input structures. This seems to be trivial, however our approach is the first uniform algorithm that is able to adapt dynamically to the data, achieving *a maximal degree of uniformity for parsing and generation under a task-oriented view.*

A performance model based on uniform processing. We are also interested in the uniform process as a basis for performance-oriented approaches like monitoring, revision, disambiguation or generation of paraphrases. We demonstrate that uniform grammatical processing increases the degree of flexibility of a natural language system, enabling it to select an utterance appropriate to the particular context. A suitable model of performance-oriented behaviour on the linguistic level is an *interleaved approach to parsing and generation.* We demonstrate that the uniform algorithm in combination with the item sharing method leads to a practical interleaved approach. In particular we present:

- **A novel method for monitoring and revision.** In situations of communication where the generation of ambiguous utterances should be avoided our method is able to compute an un-ambiguous utterance for a given semantic input. The underlying mechanism will be developed as a *chart-based incremental generate-parse-revise* strategy: substrings produced during generation are parsed to test whether they lead to some ambiguities. Detected ambiguities are handled immediately by means of revision using as much as possible of the previously computed structures. We demonstrate that such an enhanced technology enables a natural language system to reduce the risk of generating ambiguous sentences in a purposive and efficient way.

- **A novel method for the generation of paraphrases.** We also demonstrate how the same method that is used for monitoring and revision can be used for generation of paraphrases during language understanding. The idea here is that after parsing an utterance, then if this utterance has several readings, corresponding paraphrases are generated that reflect the semantic differences. The user is then asked to choose the one he intended.

The performance model takes full advantage of the uniform tabular algorithm and the item sharing method. The combination of uniform processing of reversible grammars and performance–oriented strategies are carried out in an easy and elegant way. This means that we are not only able to show – for the first time – that efficient uniform processing of reversible grammars is possible but also that systematic pursuit of uniformity in natural language processing – as followed in this thesis – achieves the necessary preconditions for a practical interleaving of parsing and generation.

Implementation The uniform tabular algorithm and the item sharing method have been fully implemented in Common Lisp and tested with several small constraint-based grammars. The incremental monitoring and revision strategy as well as the method for generation of paraphrases have also been fully implemented as straightforward extensions of the uniform tabular algorithm.

The overall structure of the model's architecture can graphically be represented as follows:

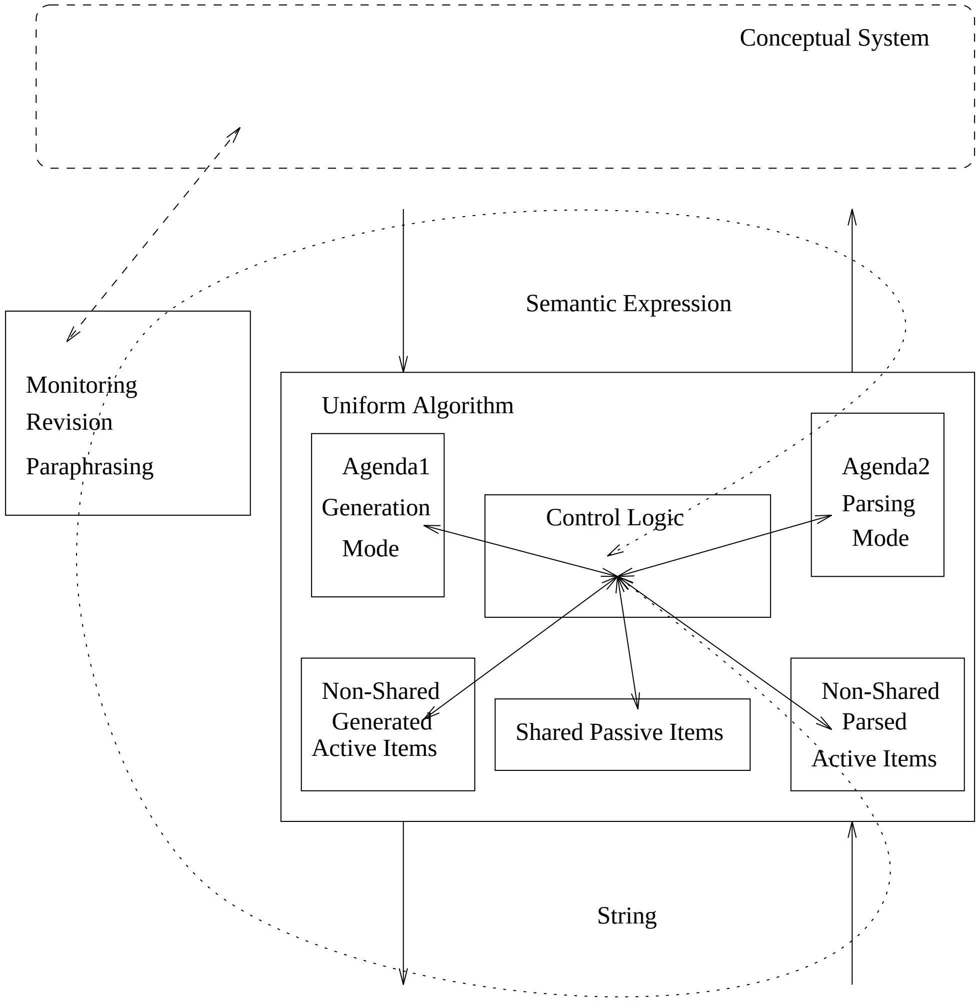


Figure 1.1: General control flow of the new model

To give a first flavor how the model works, a brief description of the flow of control follows: We assume that the conceptual system (see section 5.1 for more details) communicates via semantic expressions with the uniform algorithm. The uniform algorithm consists of a control logic (the inference rules) and an agenda mechanism. This agenda mechanism maintains two different agendas – one for generation and one for parsing. New items are first stored in one of the agendas according to a given priority (depending on whether parsing or generation is performed). Computed items are stored in different item sets, where it is assumed that passive items (which correspond to completely successfully processed substructures) are shared during generation and parsing. The reason why only shared passive items are considered will be given in section 4.15. Monitoring, revision, and paraphrasing is performed by means of specific methods. These methods are triggered (or activated) by the conceptual system but are interleaved with normal processing which is indicated by the dotted arrow.

Towards a competence-based performance model of natural language
The reversible grammar, together with the notion of derivation that underlies the uniform algorithm, constitute the grammatical competence base of a natural language system. The grammar declaratively describes the set of all possible grammatical well-formed structures of a language and the uniform algorithm is able to find all possible grammatical structures for a given input – at least potentially.

The monitoring and revision methods are designed to improve a system's performance in order to obtain an effective and flexible use of language. Thus they belong to the performance part of a natural language system.

The item sharing approach can be seen as a kind of mediator between the competence and performance methods because it is a straightforward extension of the competence base, particularly designed to support efficient interleaving of parsing and generation.

Since the competence base plays an important role for realizing these methods, the approach followed in this thesis should be seen as a step towards *a competence-based performance model* of natural language.

Clearly, additional mechanisms will be necessary for increasing the robustness, efficiency and flexibility of natural language systems. For example, machine learning methods as well as statistical or preference-based methods have to be developed and integrated into such a model in order to improve the system's performance by experience, to handle ill-formed or under-specified input, or to realize specific control strategies (see, for example, [Uszkoreit, 1991; Barnett, 1994; Samuelsson, 1994; Neumann, 1994; Wirén, 1992]).

The results presented in this thesis – especially the uniform tabular algorithm and the item sharing method – are important foundational contributions for a competence-based performance model and they should be seen as part of a long-term scientific program for achieving such a model of natural language. In the final chapter of this thesis we outline how our new model can fruitfully be combined with the above mentioned high-level performance methods.

1.2 Overview

In Chapter 2 we summarize the most important arguments for reversible natural language processing and discuss the state of the art in the area of grammar reversibility. We present a classification scheme for reversible systems and apply this scheme in the discussion of current approaches in grammar reversibility.

In Chapter 3 we present the formal and linguistic foundations on which this thesis is based. We first introduce *constraint-based grammar theories* as appropriate means for specifying reversible grammars. In these theories, the grammatical well-formedness of possible utterances is described in terms of identity constraints a linguistic structure must fulfill taking into account information of different strata (e.g., phonology, syntax and semantics) in a uniform and completely declarative way, e.g., Lexical Functional Grammar (LFG, [Bresnan, 1982]), Head-Driven Phrase Structure Grammar (HPSG, [Pollard and Sag, 1987]) and constraint-based categorial frameworks (cf. [Uszkoreit, 1986a] and [Zeevat *et al.*, 1987]).

Most important from a reversibility standpoint is that the theories only characterize *what* constraints are important during natural language use, not in what *order* they are applied. Thus they are purely declarative. Furthermore, since almost all theories assume that a natural language grammar not only describes the correct sentences of a language but also the semantic structure of grammatically well-formed sentences, they are perfectly well suited to a reversible system, because they are neutral with respect to interpretation and production.

The computational framework of our approach is based on constraint logic programming (CLP). CLP combines very well with the constraint-based view on grammar theories as well as with the deductive view of language processing, where parsing and generation are uniformly considered as proof strategies (cf. [Pereira and Warren, 1983], [Shieber, 1988] and chapter 4 of this thesis). Moreover, we show in this thesis how a tight integration of parsing and generation can be realized using CLP in an elegant and efficient way.

These aspects together makes CLP an excellent platform for combining methods from Computational Linguistics and Artificial Intelligence and hence for achieving theoretically sound and practical natural language systems.

In Chapter 4 we present the uniform algorithm. We first briefly introduce the Earley deduction framework introduced in [Pereira and Warren, 1983]. We then present a first version of the uniform tabular algorithm that makes use of a data-driven selection function, and show how it is augmented by a flexible agenda-based control regime. After illustrating how the algorithm performs parsing and generation, we extend this first version so that it can maintain structured item sets efficiently. We present a uniform indexing mechanism that can be used in the same manner for both parsing and generation. However, since we use the current portion of the input for determining the "content" of internal item sets, the item sets are ordered according to the actual structure of the input. The effect is that produced items are split into equivalence classes. The individual classes are connected by means of backward pointers, so that each item can directly be restricted to those items that belong to a particular equivalence class.

This uniform indexing mechanism is the basis of the item sharing approach whose specification is also given in that chapter. The item sharing approach is realized as an object-oriented extension of the uniform algorithm which allows us to handle different agendas and hence individual priority functions for parsing and generation.

In Chapter 5 we present a linguistic performance model which is based on the new uniform processing model and which takes full advantage of the item sharing method. We first discuss the uniform grammatical model as an *integral part* of a natural language system and its implications of the system's design. On the basis of this discussion we present a fundamental generation strategy, namely that of avoiding ambiguous output. The idea here is that the generator runs its output back through the understanding system to make sure it's unambiguous. The most advanced technology we are presenting is a *chart-based incremental generate-parse-revise* strategy. During natural language generation computed partial strings are parsed in order to test whether the partial string just produced can cause misunderstandings. If this is the case this partial string is rejected and another alternative is determined. This part of the process is called *revision*, where the parser performs the task of *monitoring* the generator's process.

We also apply this method during the understanding mode of a natural language system for the purpose of disambiguation by means of the generation of paraphrases. The idea here is that after parsing an utterance, then if this utterance is ambiguous leading to several readings, corresponding paraphrases are generated, that reflect the semantic differences. The user is then asked to choose the one he intended.

These novel techniques are realized as direct extensions of the uniform tabular algorithm, taking full advantage of the item sharing method. Thus, we are able to show that our novel approach to uniform processing leads to efficient and practical interleaving of generation and parsing.

In Chapter 6 we summarize and discuss the basic results of the thesis and outline some important future directions.

Chapter 2

Current Approaches in Reversible Systems

Natural language systems are investigated and developed in the area of cognitive science, computational linguistics, and artificial intelligence. According to [Wahlster, 1986] a system is called a *natural language system* (NLS), if

1. a subset of the input and/or output of the system is coded in natural language and

2. the processing of the input and/or generation of the output is based on knowledge about syntactic, semantic and/or pragmatic aspects of natural language.

Thus, the two primary tasks to be performed by an NLS are understanding and generating of natural language utterances on the basis of linguistic, discourse, dialog, and world knowledge.

Although the above definition does not exclude the use of other devices for communication that are more appropriate in specific situations, e.g. deictic gestures, graphics or the mix of natural language with such mediums (consider for example the systems XTRA [Allgayer *et al.*, 1989], WIP [Wahlster *et al.*, 1991], COMET [McKeown *et al.*, 1990]) the second condition rules out systems which process natural language purely as strings, e.g. text editors or compiler warnings.

During recent years there has been an increasing interest in the aspect of reversibility in natural language processing, i.e. data or programs that are shared by both natural language understanding and generation, and some significant results have emerged from theoretical linguistics, computational linguistics, and computer science.

2.1 Arguments for Reversible Natural Language Processing

A wide range of arguments can be given for reversible natural language processing, and the most important ones are summarized in the next subsections. We then present a classification scheme for systems that use reversible grammars, and discuss current approaches to uniform processing of reversible grammars.

2.1.1 Psycholinguistic Motivations

Theoretically, the assumption of same knowledge sources for both generation and understanding is essential for the common view of language as an interpersonal medium and an interface to thought [McDonald, 1987], i.e. if communication is to take place, a correspondence must be established between the mental representation of an utterance for a speaker and that for a listener. Consequently, any model of language behaviour that hypothesises only one linguistic representation is preferable to one where two different representations are used that are only applicable in one of the major modes (either understanding or generation). The fact that humans can understand meaningful utterances they have produced intentionally indicates that a subset of the internal representations used during understanding and generation must be the same. Intuitively, this must be the case at least for the mental representation of an utterance; otherwise it would be very hard to explain why humans are able to paraphrase what they have said or heard.

But there are also arguments that motivate the assumption that linguistic entities, i.e. grammatical structures are shared during understanding and generation. Garrett [1982] argues that if it would be possible to discover that understanding and generation can be modelled in substantially the same way, one might interpret this as an evidence for separating the informational structure of a language (e.g., the rule system and the lexical component of a grammar) from modality-specific aspects of the language. This *declarative* aspect of grammatical knowledge of a language is very important for illuminating the relation between grammatical theory and processing theory. For example, Fodor and Frazier [1980], argue at length that non-declarative grammars (like the ATN framework, cf. [Woods, 1986]) are inappropriate for explaining the interaction of different parsing strategies, specifically when the parser's preference for low attachment and its preference for minimal attachment are in conflict. Frazier concludes that "the claim of shared syntactic knowledge is at least coherent and consistent with available evidence concerning the mental representation of syntactic knowledge " (Frazier [1982], page 229). There is not only empirical evidence for shared syntactic knowledge but also that during understanding and generation lexical information are shared and that the actual routines used in lexical retrieval are common in both processes (see [Garrett, 1982]).

2.1.2 Engineering and Computational Motivations

A reversible grammar is also of practical relevance for the design of a natural language system. We will consider the following issues in NLS design in order to motivate the practical concerns of reversibility:

- input/output consistency of the system's language behaviour
- non–redundant knowledge representation
- portability of the system as a whole
- more flexible systems

Consistency In order to achieve *effective human-machine communication*, it is very important that a natural language system produces the same language it understands. Specifically for a dialog system a user will expect that the system will be able to understand the utterances the system has produced. For example, a user should be free to use the same words or phrases that have been introduced by the system itself. But if the system is not be able to understand its own words, it will be very difficult for the user to accept this system as an interlocutor having equal rights because the system "makes promises it cannot keep". A reversible system produces utterances only from that subset of language that it is capable of understanding. Therefore, inconsistencies in the language behaviour of the system can be avoided [Jacobs, 1988].

Non–redundancy When the linguistic knowledge of an NLS increases (especially in the case of lexical and semantic information) a system using reversible knowledge sources is less *redundant* and hence *more efficient* than a system that has to manage different sources for understanding and generation. In order to avoid inconsistencies of the language behaviour of the system as mentioned above it is necessary to update all different sources before the system can use it. The disadvantage of using different grammars and lexicons is not only that the knowledge has to be specified twice, but also that the consistency of the new knowledge with respect to the old one has to be checked in many different places. Using a reversible grammar these problems are easily avoided. For example if a user uses a word which is unknown to the system, then acquiring this word during the understanding mode of the system means that this word is automatically available for generation. Clearly, knowledge acquisition in this case presupposes that the knowledge is represented *declaratively*.

Portability and Flexibility Moreover, these aspects are very important if an NLS should be portable across different domains. In particular, if the same system is to be used as a *front–end* component in the case of understanding and as a *back–end* component in the case of generation (systems like HAM–ANS [Hoeppner *et al.*, 1983], XTRA [Allgayer *et al.*, 1989], DISCO [Uszkoreit *et al.*, 1994] are developed exactly to serve in that sense) reversibility ensures that a distinct subset of the linguistic knowledge of that system is *reusable* between different tasks. This is of great importance for the *flexibility* of a system. For example, one of the disadvantages of current generation systems is that they view the structure of linguistic knowledge only statically. If alternatives exist for a particular linguistic expression, decision points are evaluated to determine the appropriate actual utterance. It is necessary to specify corresponding decision points for all possible utterances; otherwise the choice must be performed randomly (the determination of the appropriate set of decision points is one of the sources of complexity in existing generation systems). The flexibility of such systems depends directly on the flexibility that is brought into the system via the decision points that are specified by hand during the development of a generation system (i.e. the flexibility is restricted).

When using a reversible system, structures resulting from the parsing task can be used directly during generation. This reduces the number of decision points or parameters which have to be specified during the development of an NLS which leads to more flexibility: not all necessary parameters need to be specified in the input of a generator because decision points can also be set dynamically at run-time. Consider, for example, the problem of possible ambiguity of a produced utterance. 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 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). When a reversible grammar is in use it is possible to directly compare the grammatical structures obtained during parsing and generation of an utterance. This helps to identify the relevant sources of ambiguity.

2.1.3 Adaptability to Language Use of Others

Another very important argument for the use of uniform knowledge sources is the possibility of modelling the assumption that during communication the use of language of one interlocutor is influenced by means of the language use of the others (see also [Neumann, 1991a; Neumann, 1991b]).

For example, in a uniform lexicon it does not matter whether a lexeme was accessed during parsing or generation. This means that the use of linguistic elements of the interlocutor influences the choice of lexical material during generation (if the frequency of lexemes serves as a decision criterion). This will help to choose between lexemes which are synonymous in the actual situation or when the semantic input cannot be sufficiently specified. For example, some containers can be denoted either 'cup' or 'mug' because their shape cannot be interpreted unequivocally. An appropriate choice would be to use the same lexeme that was previously used by the interlocutor (if no other information is available), in order to ensure that the same object will be denoted. In principle this is also possible for the choice between alternative syntactic structures.

This adaptation to the partner's use of language in communication is one of the sources for the fact that the global generation process of humans is flexible and efficient. Of course, adaptability is also a kind of co-operative behaviour. This is necessary, if new ideas have to be expressed for which no mutually known linguistic terms exist (e.g., during communication between experts and novices). In this case adaptability to the hearer's use of language is necessary in order to make it possible for the hearer to understand the new information.

In principle this kind of adaptability means that the input structures computed during the understanding process carry some information that can be used to parameterize the generation process.

This dynamic behaviour of a generation system will increase efficiency, too. As McDonald et al. [1987] define, one generator design is more efficient than another, if it is able to solve the same problem with fewer steps. They argue that "the key element governing the difficulty of utterance production is the degree of familiarity with the situation." The efficiency of the generation process depends on the competence and experience one has acquired for a particular situation. But to have experience in the use of linguistic objects that are adequate in a particular situation means to be adaptable.

2.1.4 Possible Arguments against Reversibility

There also seem to be certain arguments against the reversibility of understanding and generation in natural language processing, e.g., differences between active and passive language use (although this argument — as far as I know — has only been discussed for lexical material), decision-making during generation vs. hypothesis maintainance during understanding, or the differences in input structures, which cause a different traversal of the problem space.

The above discussion can be seen as an argument against this view. Clearly, it is the case that humans only use a subset of the lexical material they are able to understand. But this need not necessarily lead to the conclusion that understanding and generation are substantially different processes. Otherwise the above mentioned adaptability would be very hard to model. In a similar way [Wilks, 1991] also argues that phenomena such as active vs. passive lexical usage do not contradict a symmetrical position. Moreover he claims that "if speaking a language is to utter new and creative metaphors all the time as many researchers assert, then we can also presume that a language generator must have access to the inverse of that very same process, ..."

Some researchers (like [McDonald, 1987; Mann, 1987]) assume that the space of problems and complexity is too different such that understanding and generation could be described by the same underlying processes. Although McDonald [1987] could adopt the view that both processes could employ the same knowledge represented in the same way he states that "the two processes that draw on the knowledge cannot be the same because of the radical differences in information flow: Decision-making [during generation] is a radically different kind of process than hypothesis maintenance [during understanding] ... Understanding processes must cope with ambiguity and under-specification, problems that do not arise in generation".[1] However, as we will make clear in section 5.1, generation has also to cope with ambiguity and under-specification to be able to produce adequate utterances. Of course, one could argue that a produced utterance's ambiguity is not of relevance because the generator can assume that the hearer will be able to disambiguate it. Under this assumption generation would be less decision driven on the grammatical level than understanding, because it would be the task of the hearer to choose between alternative readings. In this view, understanding can also be interpreted as a complex decision process — the decision on understanding an utterance as best as possible.

Moreover, consider the case where a speaker wanted to talk about new ideas. He has not only to decide how to verbalize these ideas with the material already available, but he has also to hypothesize about the possible interpretations by the hearer, i.e. he has to find a way to express the new ideas such that they will be understandable (as it is usually the case in an expert–novice relationship). However, if the speaker wants to be sure that his utterance will be understood in the intended way, he has to assume that the hearer of the utterance will in principle be able to choose the inferences that he intends to be drawn. If not, it would be more or less randomly that the speaker produces exactly those structures of which analysis would lead to the intended meaning or in other words, that the hearer can reverse the speaker's computation. For example, if the speaker wanted to highlight a particular part of his ideas he can do this by choosing a specific ordering of the elements of the utterance. However, this makes sense only, if he assumes that the hearer is sensitive to linear order in exactly the same way. But then both processes should better be viewed as being symmetrical at least on the linguistic level.

In summary, reversibility has important advantages. If it is possible to develop reversible NLS which have at least the same power and functionality as non-reversible systems, such systems should be preferred. In this thesis however, we also show that the use of reversible grammars leads to *better natural language systems*.

[1] The notes written in square brackets have been added by the author.

2.2 A Classification Scheme for Reversible Systems

It is widely accepted that declarative knowledge bases are a fundamental prerequisite for achieving some degree of reversibility [Appelt, 1987]. From this point of view one can distinguish two general types of reversible natural language systems:

- systems that use reversible knowledge sources but different processes
- systems that use reversible knowledge sources as well as reversible (or *uniform*) processes

Up to now, systems that are capable of analysing and producing language fall into the first class, i.e., they use different operations for both directions (e.g., the systems HAM-ANS [Hoeppner *et al.*, 1983], XTRA [Allgayer *et al.*, 1989], CLE [Alshawi, 1992], LKP [Block, 1994], and the system DISCO [Uszkoreit *et al.*, 1994]).

Currently, it is an open question what degree of reversibility should be and can be desired (cf. [Appelt, 1987], [Mann, 1987], [McDonald, 1987], [Shieber, 1988], [Joshi, 1987], and [Neumann, 1991a]). In some areas, however, reversible processing models have been developed that are based on formalisms which are well suited for uniform representation and processing, most notably Koskenniemi's two-level model of morphology [Koskenniemi, 1984]. In computational morphology it has become the state-of-the-art to use one and the same approach for performing morphological analysis and synthesis, using both the same knowledge and the same basic mechanisms.

In the last few years the investigation of *constraint-based* grammar theories have opened up the possibility of reversible grammars and even uniform grammatical processing.

What is a reversible grammar? From a linguistic point of view a reversible grammar is the specification of grammatical knowledge in such a way that the specification is neutral with respect to its use for analysis and synthesis of well-formed expressions. In some sense, almost all modern grammatical theories consider a language as a relation between surface strings and representations of their meanings in some logical language, which are mostly referred to as *logical forms.* Parsing is then viewed as a process that assigns to a given string its possible logical forms (defined by a grammar in use) and generation is thought of as finding all valid strings that have the same given logical form.

One of the strong points of current constraint-based grammar theories is their potential to describe this relationship in a strictly declarative way, so that linguists can abstract away from specific parsing and generation strategies.

However, among researchers who are concerned with the investigation of efficient parsing and generation strategies, there is currently an active debate whether parsing and generation can really be described by one and the same efficient process. The following classification schema gives an overview of the current research directions for parsing and generation of reversible grammars (the figure 2.1 is a graphical illustration of this scheme):

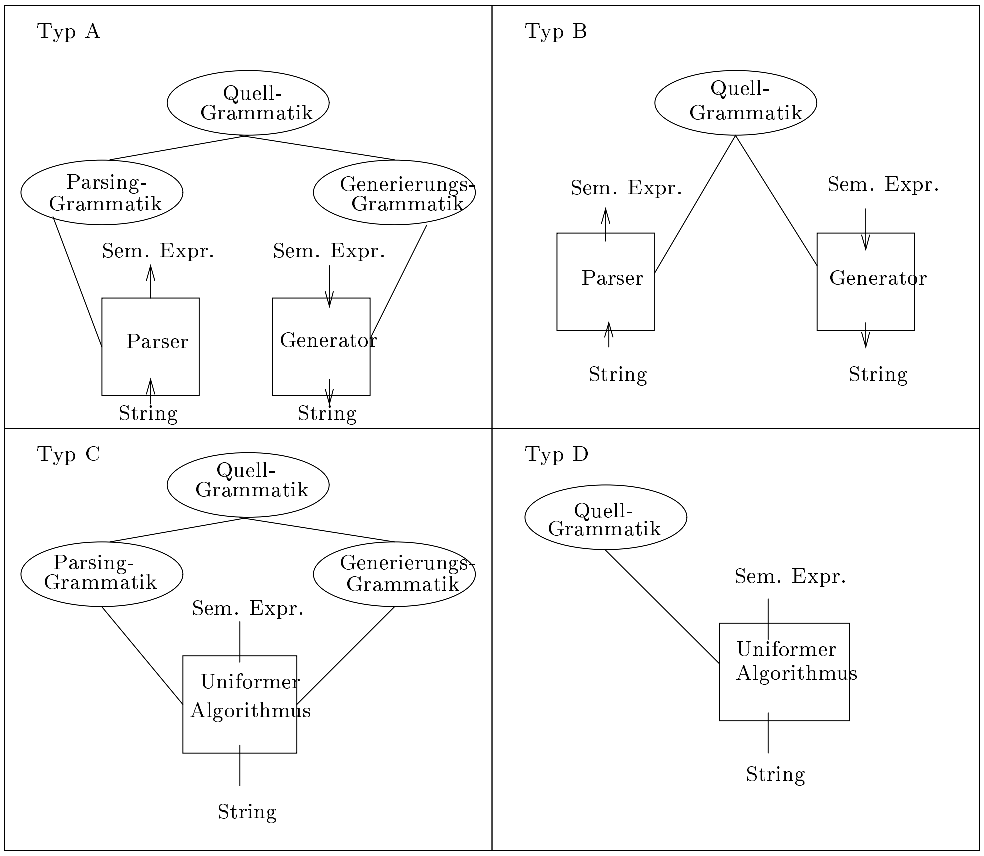


Figure 2.1: The four different types of reversible grammar approaches which are discussed in this work.

Type A Systems that use different grammars and different processes that are compiled from the same source.

Type B Systems that use a common grammar, but different processes.

Type C Systems that use different compiled grammars but a uniform process.

Type D Systems that use a common grammar, as well as a uniform process.

In systems of type A the linguistic description for parsing and generation is specified only once but is compiled into two grammars for use in the run-time mode of the system: one grammar only for parsing and one only for generation. The advantage is that the source grammar can be more 'tuned' for efficient processing during the compilation step. An example of this approach is described in [Block, 1991] where a Tomita-based LR parser is used for parsing and a variant of the semantic head–driven approach for generation. The main disadvantage of such an approach is that during run-time the same linguistic knowledge has to be stored twice which increases the amount of redundancy of the whole system.

In systems of type B the same grammar is also processed during the run-time mode by a specific parsing and generation program. The advantage of this approach is that the grammar can be changed and tested easily and that redundancy is not problematic in the case of linguistic information. An important disadvantage might be that they are too restricted in one of the directions if either the parser or generator is incomplete which in some cases can cause inconsistencies in the language behaviour of the system. An example of this approach can be found in the DISCO system [Uszkoreit *et al.*, 1994], where during parsing an Earley style parser is used and during generation a variant of the semantic head-driven algorithm.

In systems of type C specialized grammars are compiled from the same source, but are then processed by the same underlying process. For example, such an approach is followed by [Dymetman *et al.*, 1990]. Here, the basic processing regime of Prolog is used as the underlying uniform process (top-down, left-right). The grammar formalism used is lexically based, i.e., most information is contained in the lexicon (for example, subcategorization information of verbal elements). Because these kinds of grammars can cause left-recursion problems in the case of top-down processing, one goal of the compilation step is to transform left-recursive rules into equivalent non left-recursive ones. Furthermore, special "guides" are introduced during the compilation step to be able to specifically control parsing and generation. Clearly, the same disadvantage holds as for systems of type A, i.e., these systems bear a large amount of redundancy.

Systems of type D are following the most radical approach of reversibility: not only the same grammar is used but also one and the same interpreter is used for performing parsing and generation. The first detailed description of an implemented uniform architecture is due to [Shieber, 1988]. In his approach he follows the paradigm of *linguistic processing as deduction* introduced by [Pereira and Warren, 1983] in the case of parsing. Currently there are also approaches that view parsing and generation as *type rewriting* (cf. [Emele and Zajac, 1990], [Carpenter, 1992]). The advantage of a uniform approach is that inconsistency in the language behaviour can be avoided because either the restrictions are the same for both tasks or not. The main disadvantage which is often claimed, is that it is yet unclear whether and how these systems can be made equally efficient for both directions, and indeed the currently proposed approaches are problematic with respect to this issue.

There is a further important advantage of a uniform model, namely that they allow to tightly interleave parsing and generation, such that the one direction can directly re-use partial results from the other direction. In fact, in this thesis we present for the first time such an approach, which is also called *item sharing method.*

Besides the different strategies the several approaches follow, there are commonly held assumptions about the status of a reversible grammar and the parser/generator:

- Parsing and generation should be sound and complete to be able to discover (at least potentially) all possible grammatical structures of an input if the input is covered by the grammar.
- Both tasks take place independently of each other, i.e. an utterance is either generated or parsed.
- Grammatical processing can be performed without considerations of discourse.

In other words, the grammar as well as the parsing and generation processes are assumed to constitute the grammatical *competence base* of a natural language system, and — more or less explicitly — this competence base is assumed to have a modular status. Thus these approaches are consistent with important lines of research that are followed in the area of theoretical linguistics, cognitive psychology, and artificial intelligence.

When performance aspects are to be considered, additional mechanisms are necessary, for example, to realize some kind of "best-first" strategies, generation of paraphrases or possibly unambiguous utterances, or to obtain a kind of experience-based behaviour of the natural language system. If such mechanisms are to be built on the basis of the grammatical competence, then it is also an important criterion how these performance–oriented strategies can be combined or integrated with the grammatical competence base. The way in which competence and performance aspects can be combined is thus an important parameter for choosing between alternative approaches of grammar reversibility.

In this thesis we present a new model of natural language processing that is based on a strictly uniform model of the grammatical competence base and demonstrate its relevance for language performance.

We first show how efficient uniform processing is possible, by presenting a new uniform tabular algorithm for parsing and generation of constraint-based grammars. We then show how this interpreter can be hooked up with additional mechanisms, that support a tight interleaving of parsing and generation, and allow a system to perform a specific kind of self-monitoring usable in such situations where a produced utterance carries the risk of being misunderstood.

In all of these cases, we present novel methods. By describing how these several methods and their combination benefit from the uniform processing of reversible grammars, we are able to demonstrate that a uniform model (even today) is not only theoretically elegant but also of practical relevance. In this sense, our thesis can be seen as a contribution to *competence-based performance models for natural language processing.*

2.3 Problems with Existing Approaches

The purpose of this section is to discuss current approaches that describe parsing and generation under a uniform approach but explicitly taking into account efficiency and effectivity considerations. According to the classification schema specified in section 2.2 we are basically interested in approaches that belong to type D, i.e., we do not explicitly consider approaches that rely on compilation, e.g. like the one described in [Dymetman *et al.*, 1990; Strzalkowski, 1989; Block, 1991]. The main reasons of considering only truly uniform approaches in detail are that they fulfill the criterion of economy and that only for them it makes sense to develop the new item sharing approach which is necessary for achieving an efficient and practical interleaving of parsing and generation.

We also do not consider approaches, that describe alternative approaches of uniform processing, for example, the view of parsing and generation as type inference (see e.g., [Emele and Zajac, 1990]) or the use of synchronous tree-adjoining grammars (see [Shieber and Schabes, 1990]), but only under a formal or principle aspect because this would exceed the scope of this work.

2.3.1 The Uniform Architecture of Shieber

The first and most prominent attempt to specify a uniform architecture for parsing and generation has been made in [Shieber, 1988]. In particular, Shieber proposed an architecture inspired by the Earley deduction work of [Pereira and Warren, 1983] (which we will describe in section 4.1 in more detail) but which generalized that work allowing for its use in both a parsing and generation mode by instantiating only a small set of parameters. As shown in his work, the two basic inference rules of Earley deduction, namely prediction and completion can be used in the same way for parsing and generation. The advantage of using this kind of computational logic specifically for generation is that generation from grammars with recursions whose well-foundness relies on lexical information will terminate (e.g., lexicalized grammars like HPSG[Pollard and Sag, 1987], where only the verbal lexical entries carry subcategorization information). This is actually an improvement of simple top-down generation regimes like the one described in [Wedekind, 1988] or [Dymetman and Isabelle, 1988] which do not terminate in such cases. For these approaches the following rule will cause recursion problems, because the rule forces the subcategorization list SC to be expanded as long as a lexical entry that restricts the length of this list cannot be found, but the lexical entry will never be found as long as the recursion occurs:[2]

$$sign\left(\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & Tail\\ \texttt{sem:} & Sem\\ \texttt{lex:} & no\\ \texttt{v2:} & V\\ \texttt{phon:} & P_0\text{-}P\end{bmatrix}\right) \longleftarrow$$

$$sign\left(\ _{Arg}\begin{bmatrix}\texttt{phon:} & P_0\text{-}P_1\end{bmatrix}\right),\ sign\left(\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & \langle Arg|Tail\rangle\\ \texttt{sem:} & Sem\\ \texttt{v2:} & V\\ \texttt{phon:} & P_1\text{-}P\end{bmatrix}\right)$$

The problem is that the rule does not specify any finite length of the subcategorization list. Thus, this rule causes the same termination problem for generation as the well-known left-recursion problem for parsing.[3] Using an Earley style strategy, this problem can be solved, because lexical information is available immediately. However, the Earley style uniform approach described in [Shieber, 1988] has the following two major problems for generation:

- Both parsing and generation apply the *same leftmost-literal selection rule*;
- Efficient generation is possible only for grammars possessing the property of *semantic monotonicity.*

The first issue is a consequence of the fact that Shieber defines the inference rules and the meaning of items strictly based on Earley's original work on parsing algorithms [Earley, 1970]. This means that in Shieber's approach, the inference rules - prediction and completion - follow a leftmost selection strategy and items

[2]This rule states that verb phrases are built using a subcategorization list as in HPSG [Pollard and Sag, 1987] (see also [VanNoord, 1993]). Elements of the subcategorization list (bound to the feature SC) are selected one at the time by this binary verb phrase rule. The value of the SC feature is a list of signs. In this rule, the first element of the feature SC of the second daughter is equated with the first daughter of the rule. The remaining elements on the list, i.e., the tail of the list is percolated to the mother node. If a verb selects several arguments this rule can be applied iteratively. Furthermore, it is stated in this rule, that the semantics of the second daughter is identical with the semantics of the mother. The string representation of this rule expresses that the strings of the subcategorized elements of a verb are concatenated "inside-out" in reverse order.

[3]In [VanNoord, 1993] a more detailed discussion of the problems of simple top-down generators can be found.

are defined according to the position of the endpoints of the string of a completed sub-phrase. [4]

Clearly, at least in the case of a context-free backbone, parsing benefits from this characterization, but adapting this "parsing view" also for the case of generation causes at least the following two problems. Firstly, the left-to-right selection strategy is inherently more appropriate for parsing than for generation, because in the latter case this can cause a large amount of redundancies. For example, if a rule specifies that a noun phrase occurs to the left of a verb phrase then many nondeterministic possibilities for generation a noun phrase have to be explored (e.g., different cases) before the verb is generated that would restrict some of the alternatives.

The second, more important problem with a strict left-to-right selection is that generation cannot take advantage of the indexing based on string position, because the string is the output of a generator, not the input, of course. Shieber's conclusion to this problem is just to "remove the feature of tabular parsing methods such as Earley's algorithm that makes parsing reasonably efficient." ([Shieber, 1988], page 617). However, to entertain a goal-directed strategy for generation he expresses the following restriction, which directly turns our attention to the second major problem of Shieber's uniform approach: The meaning associated with an item must subsume some portion of the goal meaning. This restriction results in the *semantic monotonicity* requirement on grammars. This restriction requires that for every phrase admitted by the grammar the semantic structure of each immediate sub-phrase must subsume some portion of the semantic structure of the entire phrase.[5] But then, any item which does not subsume some part of the goal meaning can safely be ignored. For example suppose we start generation from the logical form 'passionately(love(sonny,kait))'. Furthermore, assume that the lexicon contains entries with meaning 'passionately', 'love', 'sonny' and 'kait' then a grammar is semantic monotonic if phrases with meaning 'sonny', 'kait', 'love(sonny,kait)' and 'passionately(love(sonny,kait))' will be constructed during the generation process.

Hence, for grammars that exhibit the semantic monotonicity property, Shieber's algorithm results in a complete generator. However, as stated by Shieber himself, this requirement is too strong because it does not allow generation of idiomatic expressions like 'Peter macht ein Nickerchen' with logical form 'nickerchen-machen(peter)' or expressions involving particle verbs like 'das Treffen findet morgen im DFKI statt' with logical form 'morgen(stattfinden(treffen, DFKI))'.

[4] Note that both points are not explicitly required in Pereira and Warren's original work. Hence, although Shieber's approach is a generalization of their work, in the sense that he also applies it for generation, it also could be seen as a specialization of the Earley deduction method set up in [Pereira and Warren, 1983]. Basically, this observation is the starting point of our new approach, which could be seen as real generalization of Pereira and Warren's work.

[5] Note that because of this requirement Shieber's algorithm is *coherent* in the sense that the generation process will not augment an input semantic expression.

2.3.2 The Head-driven Approach of Van Noord and Shieber et al.

In order to overcome the restriction of semantic monotonicity and to achieve a goal-directed generation strategy [Shieber *et al.*, 1989] proposed a new generation algorithm, called the *semantic head-driven generation algorithm* (SHDGA, for short) which has later been extended to parsing by [VanNoord, 1993]. A similar head-driven algorithm has been developed by [Calder *et al.*, 1989], however, restricted to use in unification-based categorial grammatical frameworks (cf. [Uszkoreit, 1986a], [Zeevat *et al.*, 1987]).

The basic motivation behind SHDGA is that generation from some semantic expression can only by directed properly if the semantic information in itself is used to direct the traversal of possible derivations. But then, a strict left-to-right processing regime cannot be used because the semantic head element (i.e., that element of the body of a rule that shares the semantics with the head) need not necessarily be the leftmost one. Consider for instance the subcategorization rule introduced in the previous paragraph. In this rule, the second element of the right-hand-side (the VP element) shares its semantics with the left-hand-side. Thus, for generation it would be more appropriate to choose and complete this element first before the first element of the right-hand-side is processed, because the head element provides important information for its 'argument'. To be able to get this kind of goal-directedness [Shieber *et al.*, 1989] proposes a *semantic-head-first selection rule* in the case of generation.

For the identification of possible semantic heads, a subset of the rules of a grammar is distinguished, called the *chain rules*, in which the semantics of some right-hand-side element is identical to the semantics of the left-hand-side. This right-hand-side element is called the *semantic head* element.[6]

All remaining rules are called *non-chain rules*. The separation of the rules of a grammar is motivated by the notion of the *pivot node*. The pivot node is defined to be the lowest node in the derivation tree such that it and all higher nodes up to the root have the same semantics. Intuitively, the pivot serves as the semantic head. Note that the pivot itself does not have a semantic head, since otherwise, it would not fit the definition. Typically, pivots will be lexical items, but in principle any un-headed phrase can serve as a pivot (e.g., idioms or conjunctions).

Using this characterization, the traversal realized by SHDGA will proceed both top-down (when processing non-chain rules) and bottom-up (when processing chain-rules) from the pivot. The algorithm then can be summarized as follows: Starting from the goal semantic expression (called the root), a rule is selected whose left-hand-side unifies with the semantics of the root. If a chain rule has been applied,

[6] Although I am using here the term introduced by Shieber et al., one should better use the term *semantic functor*, since this element need not necessarily be identical with the "syntactic head" of a phrase. For example, modifiers are often analyzed as the semantic functor of the construction they modified, whereas the modified part of the construction is the syntactic head (see also [VanNoord, 1993]).

the semantics is passed unchanged to the corresponding semantic head element, this latter non-terminal becomes the new root and the algorithm is applied recursively. If on the other hand a non-chain rule has been chosen, the algorithm is applied recursively on each element of the right-hand-side in a left-to-right scheduling. After a non-chain rule has been completed (i.e., a pivot has been identified) a bottom-up step is applied that connects the pivot to the initial root along with all intermediate nodes using a series of appropriate chain-rules. After a chain rule is chosen to move up one node in the tree being constructed, the remaining elements of the right-hand-side of this rule must be generated recursively, which is done by SHDGA using a leftmost scheduling.

To clarify the strategy by means of an example, suppose that we start generation from the semantic expression 'loves(john, mary)' using the following set of rules (using Shieber et al.'s DCG framework):

(1) sentence/decl(S) → s(finite)/S

(2) s(Head)/S → Subj, vp(Head, [Subj])/S

(3) vp(Head,Subcat)/VP → vp(Head,[Comp/LF|Subcat])/VP, Comp/LF

(4) vp(Head,Subcat)/LF → v(Head,Subcat)/LF

(5) v(finite,[np/Obj,np/Subj])/love(Subj,Obj) → [loves]

(6) np/john → [john]

(7) np/mary → [mary]

Starting with 'sentence/decl(loves(john, mary))' rule (1) is the only possible candidate to choose. To complete this rule, rule (2) has to be chosen. The body of this rule contains two elements, so there is a choice point. However, because only the second element shares its semantics with the left-hand-side this element has to be chosen and to be completed, before the first element is processed. The next possible rules to choose are (3) and (4). If (4) is considered first, further expansion of this rule will fail, because there is no lexical entry in the grammar that can unify the body of (4). In this case SHDGA backtracks to the choice point of (4) and chooses (3). Note that rule (3) extends the subcategorization list of the head element. However, applying this rule causes a recursion involving the rules (3) and (4). Now, (4) can be chosen which application will select the lexical entry (5). Now, the bottom-up step can be applied, which tries to connect the lexical entry with the initial root. During the application of this step, firstly, the semantic expression 'mary' will be realized as an object np and secondly, 'john' will be realized as the subject. The last step then will connect instantiated rule (2) with (1). The resulting string is 'john loves mary'.

The algorithm as it is described in [Shieber *et al.*, 1989] and [VanNoord, 1993] is specified as a meta-interpreter for Prolog, where Prolog's backtracking mechanism is used for maintaining alternatives. Potential nondeterminism exists in two places (see also [Gerdemann, 1991]), namely when the choice of a pivot is not determined (e.g., in the case of lexical choice) or in the case of the selection of non-chain rules for phrasal pivots. This latter case is crucial because for non-chain rules the order of processing the elements of the right-hand side of a non-chain rule is not determined so that SHDGA applies itself recursively in a simple left-to-right order.

The most serious problem of SHDGA, however, is that the top-down processing of non-chain rules can lead to non-termination (see also [Gerdemann, 1991]). For example consider the following rule:[7]

nbar/N $\rightarrow$ nbar/N, sbar/N

This rule can effect nontermination, if the generator chooses the second element of the body first. Clearly, if we could define a *head-driven* Earley style algorithm, these problems would easily be avoided, and in fact, this has been done by [Gerdemann, 1991], which we will discuss in the next paragraph.

However, before we move to Gerdemann's approach we have some remarks on *head-driven* parsing, preliminary to discussing issues of uniformity underlying SHDGA.

For parsing, algorithms have also been developed, that favour a head-driven processing regime, most notably [Kay, 1989; VanNoord, 1993]. Both approaches are modifications of left-corner parsers [Matsumoto *et al.*, 1983], such that instead of choosing the leftmost element of the body of a rule the parser selects the lexical head first (actually, the parser selects a lexical entry and continues to prove that this lexical entry indeed is the head of the goal, by selecting a rule of which this lexical entry can be the head). The other rules are then parsed recursively, and the result constitutes a slightly larger head. This process can be applied iteratively, until the head dominates all words of a string. Because the bottom-up approach is head-oriented, these algorithms are also denoted as *head-corner parsers*.

The algorithm described by [VanNoord, 1993] is closely related to SHDGA, so that both approaches can be seen as a uniform framework for reversible grammars. Although both algorithms are very similar, Van Noord does not try to combine both, in order to obtain one uniform parameterized approach. Furthermore, it is not clear whether one and the same grammar can be used efficiently in both directions without the need of compilation. In fact, [Martinovic and Strzalkowski, 1992] give an example of a grammar that can be used efficiently for parsing, but which causes a nondeterministic overhead when directly processed by SHDGA. The problem they focused on is that the left-to-right selection strategy might in the case of non-head

[7] This example is taken from [Shieber *et al.*, 1989] where they discuss the problem in a footnote on page 10.

elements select an element which is not "ready", i.e., whose semantics is still uninstantiated. The only way to avoid such problems for SHDGAwould be to rewrite the underlying grammar, so that the choice of the most instantiated literal on the right-hand-side of a rule is forced. However, this can cause a "ping-pong" effect, because now, it is not guaranteed that the parsing mode will handle this grammar in the same efficient manner as before.

2.3.3 Gerdemann's Earley Style Processing Scheme

In his thesis [Gerdemann, 1991] introduced algorithms for parsing and generation of constraint-based grammars, where both are specific instantiations of an Earley style approach. The Earley style parsing algorithm is very similar to the one presented in [Shieber, 1988] and [Shieber, 1989] and differs basically in improvements concerning the use of efficient application of subsumption checks and in being able to handle variable length lexical items (including traces). For the purpose of the current discussion we will therefore focus our attention on his adaption of the Earley style generator, before discussing its degree of uniformity.

The major aspect of Gerdemann's generation approach is the use of a *head-driven Earley generation* algorithm, i.e., instead of following a left-to-right selection rule, he adapts the head-first selection rule introduced in SHDGA. Using this control regime also in the case of an Earley style approach he shows that he can achieve the same kind of goal-directness as SHDGA but avoids the kind of nondeterminism and recursion loops, for which SHDGA comes into trouble. Furthermore, he shows how to handle semantic non-monotonicity and how to use semantic information for indexing already derived structures, so that in his approach the generator can use an already generated phrase in more than one place (if necessary). [8]

The first issue emphasized by Gerdemann is that the semantic monotonicity requirement is not necessary for an Earley type generation algorithm. As already discussed in paragraph 2.3.1, in order to be able to use the same generated phrases on different string positions, Shieber eliminates the string position indexing, by collapsing all of the item sets to one. However, as argued by Shieber himself "The generation behaviour exhibited is therefore not goal-directed;" ([Shieber, 1988], page

[8] It would be possible to avoid this kind of processing overhead also for SHDGA, using so-called memoization techniques (for technical aspects of memoization see e.g., [Norvig, 1992; Pereira and Shieber, 1987]). In fact, [VanNoord, 1993] discusses the use of memoization for semantic-head driven generators. Memoization can only be performed in combination with subsumption tests. However, as argued by Van Noord, "Even though the overhead involved in the implementation of such memo-relations is still considerable, it turns out that for many grammars the head-driven generator is more efficient if implemented as a memo-relation ... by a factor 4 for typical grammars." ([VanNoord, 1993], pages 93-94). It is interesting to note that the same author (when motivating his own approach) emphasized as a disadvantage of Shieber's Earley style generator that "... the necessary subsumption checks (for example to check whether a result already is present in the chart) lead to much processing overhead." ([VanNoord, 1993], page 80).

617). In order to achieve goal-directedness, he introduces the semantic monotonicity condition (already discussed in 2.3.1). But the goal directedness of Earley's algorithm does not come from the state sets, it comes from top-down prediction. Clearly, if a head-first selection rule is chosen (as known from SHDGA), a goal-directed behaviour can be obtained comparable to that of SHDGA but without the requirement of semantic monotonicity. Of course, this implies that the restriction function used in the prediction step should ignore semantic information. Gerdemann concludes that "If this non-goal directedness due to restriction is ignored, it is not at all clear why a generation algorithm that collapsed all states into one state set would lose the normal goal directedness that comes from prediction." ([Gerdemann, 1991], page 78).

In order to take full advantage of an Earley style strategy for generation, one has to face the problem of avoiding re-computation of phrases that can be placed at different string positions, i.e., to avoid backtracking in those cases. The basic idea followed by Gerdemann is to retain the state sets but to modify them so that whenever it happens that the generator starts to generate a duplicate phrase at a new string position, the chart will be readjusted so that it appears that the duplicate phrase is being generated in the same string position as the original phrase ([Gerdemann, 1991], page 80).

In order to make this modification, he introduced a global list (called GRD), which has to be maintained by the generator. This global structure consists of a list of all restrictors[9] that have been used so far to make predictions with. Note that Gerdemann requires that restrictors should contain all (local) semantic information, so that this list also indicates which semantic information has been involved in making a prediction. Each entry on this global list is of the form [RD,F,C], where RD was used to make the prediction, F is the number of the state set in which the prediction has been made, and C is a list of all the complete phrases that have so far been generated from RD.

The list GRD is now used in the predictor and completor step as follows: If in the predictor an RD is created that is subsumed by an RD already in GRD, no new predictions are made, since any such predictions would be subsumed by predictions made earlier. If the subsuming RD was used in state F, then the current state will be *moved* to the Fth state set because any completions that are found for the previously made predictions will be linked back to the Fth state set. The effect of this moving operation is that it now appears that the duplicate phrase being predicted is actually predicted to start at the same string position. But now, by moving the current state back to the state set of a previously used RD, any completion for the previous state (recorded in C) can also be used as completion for the current state.

There are two problematic aspects of Gerdemann's approach with respect to the degree of uniformity. Firstly, it seems to be the case that a grammar has to be com-

[9]The reader not familiar with the notation of restriction should consult section 4.1.

piled into a specific parsing and generation grammar since in his implementation of the algorithm he followed a left-to-right scheduling (see Appendix C of [Gerdemann, 1991]). Secondly (and more important), the modifications necessary in order to maintain the global list GRD to avoid generation of duplicate phrases "... have fundamentally altered Earley's algorithm." ([Gerdemann, 1991], page 89). Since, the GRD list is only used during generation, the parsing and generation algorithms are substantially different. Both aspects together strongly weaken the uniform character of Gerdemann's approach.

2.3.4 Summary

We can summarize the discussion of current approaches in grammar uniformity as follows: Based on the pioneering work of [Pereira and Warren, 1983] who demonstrate that parsing can be modelled as deduction introducing an Earley deduction method, [Shieber, 1988] generalizes their work to be applicable also for generation. However, Shieber's approach depends too strongly on a parsing view, so that in his approach the generation mode is less efficient than the parsing mode. To overcome this problem, [Shieber *et al.*, 1989] introduce a new generation algorithm, which exhibits goal-directed generation behaviour by introducing a semantic-head first selection strategy, instead of the left-to-right one used by [Shieber, 1988]. However, applying this approach also for parsing, it seemed to be the case that parsing now, loses efficiency. Furthermore, the algorithm of [Shieber *et al.*, 1989] has to face problems of nondeterminism and nontermination in the case of the top-down processing of non-chain rules, basically because they use the simple backtracking mechanism known from Prolog. To overcome these problems, Gerdemann adapts the idea of using a head-driven strategy in the case of generation but "backtracks" to Shieber's Earley style approach. He then shows that he obtains the same goal-directedness as Shieber et al., but without the problems of unnecessary re-computation and by avoiding recursion loops. However, the generator he developed on top of Earley's approach fundamentally altered this algorithm (which Gerdemann uses for parsing) and hence loses a certain degree of uniformity.

The uniform approach that we are going to describe in this thesis is more strictly based on Pereira and Warren's approach. It is the first real generalization of their work with respect to parsing and generation, because our algorithm is able to adopt itself for the specific tasks at hand. Thus we can take advantage of a left-to-right selection strategy for parsing and a semantic head-driven selection strategy for generation using the same algorithm. Furthermore, because we use uniform indexing techniques we can define prediction and completion in a truly uniform way, thus avoiding the use of global structures like the one defined in Gerdemann's approach. Hence, our approach seems to be the most general view on parsing and generation which combines the advantages of most of the current approaches in one place. Furthermore, because we can choose any selection function, we easily can take advantage

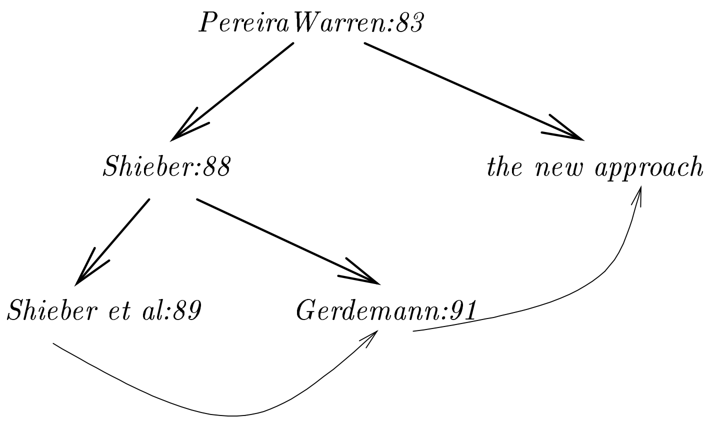


Figure 2.2: A summary of the approaches discussed in relationship to the new approach. The arrows indicate the relationship of most direct influence.

of a strict *data-driven* view. Thus, we can put in perspective our new approach using the figure 2.2.

Finally it is important to note that in none of the current uniform approaches neither interleaving of parsing and generation nor the possibility of sharing items between parsing and generation as the basis of a performance-oriented model has been discussed. Since we demonstrate that such a method is necessary in order to interleave parsing and generation in a practical way our approach has superior power than all of the todays uniform approaches.

Chapter 3

Linguistic and Formal Foundations

This chapter is concerned with the linguistic and formal foundations used for the competence-based performance model.

We first introduce *constraint-based grammar theories* as appropriate means for specifying reversible grammars. In these theories, the grammatical well-formedness of possible utterances is described in terms of identity constraints a linguistic structure must fulfill taking into account information of different strata (e.g., phonology, syntax and semantics) in a uniform and completely declarative way, e.g., Lexical Functional Grammar (LFG, [Bresnan, 1982]), Head-Driven Phrase Structure Grammar (HPSG, [Pollard and Sag, 1987]) and constraint-based categorial frameworks (cf. [Uszkoreit, 1986a] and [Zeevat *et al.*, 1987]).

Most important from a reversibility standpoint is that the theories only characterize *what* constraints are important during natural language use, not in what *order* they are applied. Thus they are purely declarative. Furthermore, since almost all theories assume that a natural language grammar not only describes the correct sentences of a language but also the semantic structure of grammatically well-formed sentences, they are perfectly well suited to a reversible system, because they are neutral with respect to interpretation and production.

The computational framework of our approach is based on constraint logic programming (CLP). CLP combines very well with the deductive view of language processing where parsing and generation are uniformly considered as proof strategies (cf. [Pereira and Warren, 1983], [Shieber, 1988] and chapter 4 of this thesis) as well as with the constraint-based view on current grammar theories. Moreover, we show in this thesis how a tight integration of parsing and generation can be realized using CLP in an elegant and efficient way.

These aspects together makes CLP an excellent platform for combining methods from Computational Linguistics and Artificial Intelligence and hence for achieving theoretically sound and practical natural language systems.

The rest of the chapter is organized as follows. In section 3.1 we informally present the basic ideas shared by all modern constraint-based grammar theories. Section 3.2 presents the formalism of [Höhfeld and Smolka, 1988] which is a general characterization of such constraint-based formalisms and an actual constraint language (called $\mathcal{L}$) for representing linguistic structures. Although we have chosen a simple constraint language in order to highlight the new results in a clean but simple way, the generalization of the scheme guarantees that the results of this thesis also carries over for more complex constraint languages. In section 3.3 we show how the formalism is used for writing grammars and in section 3.4 we introduce the constraint-based view of natural language parsing and generation and discuss the relationship between ambiguities and paraphrases.

3.1 Constraint-based Grammars

Since the last decade a family of linguistic theories known under the term *constraint-based grammar theories* play an important role within the field of natural language processing. One of the reasons for their importance is that they are *declarative*, i.e., are able to express grammatical knowledge without regard to any specific use. Since they also integrate information from different levels of linguistic description uniformly they are very well suited as the linguistic base of a reversible system. In this sense, constraint-based grammars are reversible.

The common design feature of these formalisms is that they model linguistic entities (e.g., words and phrases) "... as *partial information structures*, which mutually constrain possible collocations of phonological structure, syntactic structure, semantic content, and contextual factors in actual linguistic situations. Such objects are in essence data structures which specify values for attributes; their capability to bear information of non-trivial complexity arises from their potential for recursive embedding (the value of some attribute may be another information structure with internal structure of its own) and structure-sharing (one and and the same structure may occur as the value of distinct attributes in a larger structure)" [Pollard and Sag, 1987], page 7. Since, these informational structures are viewed as partial, they are used to represent only necessary restrictions or *constraints* of such properties a linguistic object must fulfill; hence the name constraint-based theories. To give an example, consider the following simplified attribute-value representation (also known as *feature structure*) in the commonly used feature-matrix notation, which is intended to represent the (partial) information about the utterance "peter cries":

$$\begin{bmatrix} \texttt{cat} & \textit{sentence} \\ \texttt{phon} & \langle \textit{peter}, \textit{cries} \rangle \\ \texttt{syntax} & \ldots \\ \texttt{dtrs} & \left\langle \begin{bmatrix} \texttt{cat} & \textit{noun} \\ \texttt{phon} & \textit{peter} \\ \texttt{syntax} & \begin{bmatrix} \texttt{agr} & \begin{bmatrix} \texttt{per} & \textit{3} \\ \texttt{num} & \textit{sg} \end{bmatrix} \end{bmatrix} \\ \texttt{semantics} & {}_{\textit{Arg}}\begin{bmatrix} \texttt{rel} & \textit{the-peter'} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \texttt{cat} & \textit{verb} \\ \texttt{phon} & \textit{cries} \\ \texttt{syntax} & \begin{bmatrix} \texttt{agr} & \begin{bmatrix} \texttt{per} & \textit{3} \\ \texttt{num} & \textit{sg} \end{bmatrix} \end{bmatrix} \\ \texttt{semantics} & {}_{\textit{Sem}}\begin{bmatrix} \texttt{rel} & \textit{cry} \\ \texttt{arg} & \textit{Arg} \end{bmatrix} \end{bmatrix} \right\rangle \\ \texttt{semantics} & \textit{Sem} \end{bmatrix}$$

This structure expresses that the utterance "peter cries" is a linguistic object of class sentence, whose phonological representation is $\langle \textit{peter}, \textit{cries} \rangle$, which has been formed from two objects, one for the utterance "peter" and one for "cries" and that the semantic content is the same as that of "cries" which by itself is built by combining the semantic structure of 'cry' and 'the-peter'. Clearly, this data structure is only a model of the 'real' linguistic object. However, it characterizes relevant properties of the information the 'real' object conveys.

The underlying assumption of all constraint-based theories is that in the actual utterance situation the specification of linguistic objects is built in a monotonic way independently of the specific processing task at hand (e.g., parsing, generation) from different sources, including lexicon and grammar rules, as well as from language universal and language specific principles. Building larger objects from smaller ones is assumed to be performed by some sort of constraint-solving whose task is to collect all constraints of the various attending objects, so that the resulting structure describes a consistent, i.e., grammatical linguistic object.[1]

Most important from a reversibility standpoint is that the theories only characterize *what* constraints are important during natural language use, not in what *order* they are applied. Thus they are purely declarative. Furthermore, since almost all theories assume that a natural language grammar not only describes the correct sentences of a language but also the semantic structure of grammatically well-formed sentences, they are perfectly well suited to a reversible system, because they are neutral with respect to interpretation and production.

In the last few years constraint-based formalisms have undergone a rigorous formal investigation (consider for example [Shieber, 1989; Smolka, 1988; Smolka, 1992]). This has led to a general characterization of constraint-based formalisms where feature structures are considered to constitute a semantic domain and constraints are

[1] In the beginning of their formalization, unification was the predominant constraint solving mechanism; hence they often referred to as *unification-based grammars*.

considered syntactic representations of such 'semantic structures'. This logical view has several advantages. On the one hand, it has been possible to properly incorporate concepts like disjunction or negation as part of the (syntactic) constraint language and to interpret them relative to a given domain of feature structures (usually defined as graph-like or tree-like structures). On the other hand it has been possible to combine constraint-based formalisms with logic programming, which fits into a new research area known under the term *constraint logic programming* (CLP) [Jaffar and Lassez, 1987]. This enables us to extend the view of natural language as deduction for essentially arbitrary constraint-based formalisms.

A general characterization of constraint logic programming is given in [Höhfeld and Smolka, 1988]. They show that the nice properties of logic programs extend to definite clause specifications over arbitrary constraint languages.

We will use the scheme of [Höhfeld and Smolka, 1988] as our underlying formal language. We therefore first summarize the most important definitions and statements given there. We then specify a concrete constraint language also called $\mathcal{L}$ which is based on the definitions of [Smolka, 1992] and [VanNoord, 1993]. This constraint language will be used as the central data structure for representing linguistic objects. The relational extensions provided by $\mathcal{R}(\mathcal{L})$ then serve as the basis for representing grammatical rules, i.e., complex compositional entities. The use of the presented formalism in writing grammars is illustrated in section 3.3.

3.2 Constraint Logic Programming

In constraint logic programs basic components of a problem are stated as constraints (i.e., the structure of the objects in question) and the problem as a whole is represented by putting the various constraints together by means of rules (basically by means of definite clauses). For example the following definite clause grammar (using the constraint language $\mathcal{L}$ defined in section 3.2.2)

$$
\begin{array}{l}
sign(X_0) \leftarrow \\
\quad sign(X_1), \\
\quad sign(X_2), \\
\quad X_0\ syn\ cat \doteq s, \\
\quad X_1\ syn\ cat \doteq np, \\
\quad X_2\ syn\ cat \doteq vp, \\
\quad X_1\ syn\ agr \doteq X_2\ syn\ agr
\end{array}
$$

expresses that for a linguistic object to be classified as an S phrase it must be composed of an object classified as an NP and by an object classified as a VP and the agreement information between NP and VP must be the same. All objects that fulfill at least these constraints are members of S objects. Note that there is no ordering presupposed for NP and VP as is the case for unification-based formalisms that rely on a context-free backbone (e.g., [Shieber *et al.*, 1983]). If such a restriction is required additional constraints have to be added to the rule, for instance that substrings have to be combined by concatenation.

Since the constraints in the example above only specify necessary conditions for an object of class S, they express partial information. This is very important for linguistic processing (or other knowledge-based reasoning), because in general we have only partial information about the world we want to reason with.

Processing of such specifications is then based upon constraint solving and the logic programming paradigm. Because unification is but a special case of constraint solving, constraint logic programs have superior expressive power.

In [Höhfeld and Smolka, 1988] a general relational extension is made for arbitrary constraint languages. Given a constraint language $\mathcal{L}$ and a set $\mathcal{R}$ of relation symbols, $\mathcal{L}$ is extended conservatively to a constraint language $\mathcal{R}(\mathcal{L})$ providing for relational atoms, the propositional connectives, and quantification. In particular, they show how the properties of logic programming carry over to a whole range of constraint-based formalisms, by abstracting away from the actual constraint language in use.

3.2.1 Constraint Languages and Relational Extensions

Constraint Language To start with we give an abstract definition of a constraint language. According to [Höhfeld and Smolka, 1988] a constraint ϕ is some piece of syntax constraining the values of the variables occurring in it, i.e., which denotes a set of assignments for these variables relative to a given interpretation.

Definition 1 (Constraint Language) *A constraint language is a tuple* $\langle VAR, CON, \mathcal{V}, INT \rangle$ *such that:*

1. *VAR is a decidable, infinite set of variables*
2. *CON is a decidable set of constraints*
3. $\mathcal{V}$ *is a computable function* $CON \rightarrow 2^{VAR}$ *that assigns to every constraint* ϕ *a finite set* $\mathcal{V}\phi$ *of variables, called the variables constrained by* ϕ
4. *INT is a nonempty set of interpretations, where every interpretation* $\mathcal{I} \in INT$ *consists of a nonempty set* $\mathcal{D}^{\mathcal{I}}$*, called the domain of* $\mathcal{I}$*, and a solution mapping* $[\![.]\!]^{\mathcal{I}}$ *such that:*

   (4.1) *an* $\mathcal{I}$*-assignment is a mapping* $VAR \rightarrow \mathcal{D}^{\mathcal{I}}$*, and* $\mathcal{ASS}^{\mathcal{I}}$ *denotes the set of all* $\mathcal{I}$*-assignments*

   (4.2) $[\![.]\!]^{\mathcal{I}}$ *is a function mapping every constraint* ϕ *to a set* $[\![\phi]\!]^{\mathcal{I}}$ *of* $\mathcal{I}$*-assignments, where the* $\mathcal{I}$*-assignments in* $[\![\phi]\!]^{\mathcal{I}}$ *are called the* solutions of ϕ in $\mathcal{I}$.

   (4.3) *a constraint* ϕ *constrains only the variables* $\mathcal{V}\phi$*, that is, if* $\alpha \in [\![\phi]\!]^{\mathcal{I}}$ *and* β *is an* $\mathcal{I}$*-assignment that agrees with* α *on* $\mathcal{V}\phi$*, then* $\beta \in [\![\phi]\!]^{\mathcal{I}}$.

The following definitions are all made with respect to some given constraint language.

A constraint ϕ is *satisfiable* if there exists at least one interpretation in which ϕ has a solution. A constraint ϕ is *valid* in an interpretation $\mathcal{I}$ if every $\mathcal{I}$-assignment is a solution of ϕ in $\mathcal{I}$, i.e., if $[\![\phi]\!]^{\mathcal{I}} = \mathcal{ASS}^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* a constraint ϕ if ϕ is valid in $\mathcal{I}$. An interpretation is a *model* of a set Φ of constraints if it satisfies every constraint in Φ.

The *subsumption* preordering on sets of constraints and the corresponding *equivalence relation* is defined as follows (following the notation of [Dörre, 1993]):

Definition 2 (Subsumption, Equivalence)

$$\begin{array}{lll} \phi \sqsubseteq \psi \ (\phi \textit{ subsumes } \psi) & :\Longleftrightarrow & [\![\psi]\!]^{\mathcal{I}} \subseteq [\![\phi]\!]^{\mathcal{I}} \textit{ for all } \mathcal{I} \\ \phi \sim \psi & :\Longleftrightarrow & \phi \sqsubseteq \psi \wedge \psi \sqsubseteq \phi \end{array}$$

A *variable renaming* is a bijection VAR $\rightarrow$ VAR that is the identity except for finitely many exceptions. If ρ is a renaming, a constraint ϕ' is called a ρ*-variant* of a constraint ϕ if

$$\mathcal{V}\phi' = \rho(\mathcal{V}\phi) \text{ and } [\![\phi]\!]^{\mathcal{I}} = [\![\phi']\!]^{\mathcal{I}}\rho \ := \ \{\alpha\rho \mid \alpha \in [\![\phi']\!]^{\mathcal{I}}\}$$

for every interpretation $\mathcal{I}$ ($\alpha\rho$ denotes the functional composition of both functions). A constraint ϕ' is called a *variant* of a constraint ϕ if there exists a renaming ρ such that ϕ' is a ρ-variant of ϕ. Note that renaming is homomorphic with respect to the subsumption relation, thus it does not affect the subsumption ordering of constraints (see proposition 2.2 in [Höhfeld and Smolka, 1988]).

A constraint language is closed under renaming if every constraint ϕ has a ρ-variant for every renaming ρ. A constraint is *closed under intersection* if for every two constraint ϕ and ϕ' there exists a constraint ψ such that $[\![\phi]\!]^{\mathcal{I}} \cap [\![\phi']\!]^{\mathcal{I}} = [\![\psi]\!]^{\mathcal{I}}$ for every interpretation $\mathcal{I}$.

A constraint language is decidable if the satisfiability of its constraints is decidable. In section 3.2.2 we present a decidable constraint language.

A constraint language is *compact* if for every set of constraints Φ holds: Φ is satisfiable iff every finite subset of Φ is satisfiable.

Before we present in section 3.2.2 the constraint language to be used in this thesis we present the relational extension of constraint languages.

Relational Extension As already said CLP consists of constraints and rules for combining various constraints. In the scheme of [Höhfeld and Smolka, 1988] this is done by adding a set $\mathcal{R}$ of relation symbols to a constraint language $\mathcal{L}$ which yields a constraint language $\mathcal{R}(\mathcal{L})$ providing for relational atoms, the propositional connectives and quantification. The restriction to definite clauses then allows the adoption on well-known standard logic programming concepts like SLD-resolution, which defines an operational semantics for $\mathcal{R}(\mathcal{L})$.

Definition 3 (Relational Extension) *The relational extension $\mathcal{R}(\mathcal{L})$ of a constraint language $\mathcal{L}$ with respect to a decidable set $\mathcal{R}$ of relational symbols is as follows:*

(1) the variables of $\mathcal{R}(\mathcal{L})$ are the variables of $\mathcal{L}$

(2) the constraints of $\mathcal{R}(\mathcal{L})$ are inductively defined as follows:

(2.1) every constraint of $\mathcal{L}$ is a constraint of $\mathcal{R}(\mathcal{L})$

(2.2) if r is a relation symbol of $\mathcal{R}$ and $\vec{x}$ is a tuple of pairwise distinct variables, then the relational atom $r(\vec{x})$ is a constraint of $\mathcal{R}(\mathcal{L})$, provided the tuple $\vec{x}$ has as many elements as r has arguments

(2.3) the empty conjunction $\emptyset$ is a constraint of $\mathcal{R}(\mathcal{L})$; furthermore, if ϕ and ψ are constrains of $\mathcal{R}(\mathcal{L})$, then the conjunction ϕ,ψ and the implication $\phi \rightarrow \psi$ are constraints of $\mathcal{R}(\mathcal{L})$

(2.4) if x is a variable and ϕ is a constraint of $\mathcal{R}(\mathcal{L})$, then the existential quantification $\exists x.\phi$ is a constraint of $\mathcal{R}(\mathcal{L})$

(3) the variables constrained by a constraint of $\mathcal{R}(\mathcal{L})$ are defined inductively as follows: if ϕ is an $\mathcal{L}$-constraint then $\mathcal{V}\phi$ are defined as in $\mathcal{L}$; $\mathcal{V}(r(x_1, \ldots, x_n))$ $:= \{x_1, \ldots, x_n\}$; $\mathcal{V}\emptyset := \emptyset$; $\mathcal{V}(\phi, \psi) := \mathcal{V}\phi \cup \mathcal{V}\psi$; $\mathcal{V}(\phi \rightarrow \psi) := \mathcal{V}\phi \cup \mathcal{V}\psi$; $\mathcal{V}(\exists x.\phi)$ $:= \mathcal{V}\phi$ - $\{x\}$

(4) an interpretation $\mathcal{A}$ of $\mathcal{R}(\mathcal{L})$ is obtained from an $\mathcal{L}$-interpretation $\mathcal{I}$ by choosing for every relation symbol $r \in \mathcal{R}$ a relation $r^{\mathcal{A}}$ on $\mathcal{D}^{\mathcal{I}}$ taking the right number of arguments, and by defining:

(4.1) $\mathcal{D}^{\mathcal{A}} := \mathcal{D}^{\mathcal{I}}$

(4.2) $[\![\phi]\!]^{\mathcal{A}} := [\![\phi]\!]^{\mathcal{I}}$, *if ϕ is an $\mathcal{L}$-constraint*

(4.3) $[\![r(\vec{x})]\!]^{\mathcal{A}} := \{\alpha \in \mathcal{ASS}^{\mathcal{A}} | \alpha(\vec{x}) \in r^{\mathcal{A}}\}$

(4.4) $[\![\emptyset]\!]^{\mathcal{A}} := \mathcal{ASS}^{\mathcal{A}}, \quad [\![\phi, \psi]\!]^{\mathcal{A}} := [\![\phi]\!]^{\mathcal{A}} \cap [\![\psi]\!]^{\mathcal{A}}$

(4.5) $[\![\phi \rightarrow \psi]\!]^{\mathcal{A}} := (\mathcal{ASS}^{\mathcal{A}} - [\![\phi]\!]^{\mathcal{A}}) \cup [\![\psi]\!]^{\mathcal{A}}$

(4.6) $[\![\exists x.\phi]\!]^{\mathcal{A}} := \{\alpha \in \mathcal{ASS}^{\mathcal{A}} | \exists \beta \in [\![\phi]\!]^{\mathcal{A}}\ \forall y \in \mathcal{V}\phi . y = x \ \vee \ \beta(y) = \alpha(y)\}$.

Since $\mathcal{R}(\mathcal{L})$ is a constraint language, all definitions we have made for constraint languages apply to $\mathcal{R}(\mathcal{L})$. Thus the notion of a constraint language can be applied iteratively.

Definite clauses A *definite clause* is an $\mathcal{R}(\mathcal{L})$-constraint of the form:

$$p_1, p_2, \ldots, p_n, \phi \rightarrow q$$

where $n \geq 0$, $p_1, p_2, \ldots p_n$ and q are atoms and ψ is an $\mathcal{L}$-constraint. We call q the head of a clause and $p_1, p_2, \ldots p_n$ its body. We may write a clause as $q \leftarrow p_1, \ldots, p_n, \phi$ or simply as $q \leftarrow p$. In case the head and the body of a clause is empty, we call the clause an *empty clause*. In general, an empty clause is of form

$$\leftarrow \phi$$

A *definite clause specification* is a set of definite clauses. Höhfeld and Smolka show that important properties of conventional logic programs extend to definite clause specifications, especially the existence of a unique minimal model for each interpretation in $\mathcal{L}$.

A *goal* is a possibly empty conjunction of $\mathcal{R}(\mathcal{L})$-atoms and an $\mathcal{L}$-constraint, written as:

$$\leftarrow p_1, \ldots, p_n, \phi$$

that is, a clause with an empty head (or consequent).

An *S-answer* to a goal with respect to a given definite specification S is a satisfiable constraint ψ, such that $\psi \rightarrow p_1, \ldots, p_n, \phi$ is valid for every minimal model of S.

Operational semantics Höhfeld and Smolka provide a generalization of the SLD-resolution method known from standard logic programming (cf. [Lloyd, 1987]) to definite clauses in $\mathcal{R}(\mathcal{L})$. The key result of this generalization is that most important results from conventional logic programming carry over for definite clause specifications using arbitrary constraint languages.

The fundamental inference rule for definite clauses in $\mathcal{R}(\mathcal{L})$ is the following *goal reduction* rule (using a slightly different notation from that given in [Höhfeld and Smolka, 1988])

$$p_1, \ldots, p(\vec{x}), \ldots p_n, \phi \Longrightarrow p_1, \ldots, q_1, \ldots, q_m, \ldots, p_n, \phi, \psi$$

where $p(\vec{x})$ is the selected element of a goal and

$$p(\vec{x}) \leftarrow q_1, \ldots, q_m, \psi$$

is a variant of a clause of a definite clause specification S, such that the variables in the clause do not occur in the original goal, except for the variables explicitly mentioned, which means that the variant must be variable-disjoint from the original goal (the antecedent part of the goal reduction rule). A variant of a clause can be obtained by renaming the variables of the clause.

We also say that the new goal $p_1, \ldots, q_1, \ldots, q_m, \ldots p_n, \phi, \psi$ is a *derived* form and call it the *resolvent* of the goal $p_1, \ldots, p(\vec{x}), \ldots, p_n, \phi$ and $p(\vec{x}) \leftarrow q_1, \ldots, q_m, \psi$. Höhfeld and Smolka show that goal reduction is a *sound and complete basic inference rule* for definite clause specifications in $\mathcal{R}(\mathcal{L})$.

Note that the inference rule is only applied in case the resulting constraint ϕ, ψ is satisfiable, since otherwise we would not be able to find some answer. Therefore, we will make use of the following optimization known from conventional logic programming and proven by [Höhfeld and Smolka, 1988] for the general case:

Immediately after a goal reduction step, it is checked, whether the resulting constraint ϕ, ψ is satisfiable through a *constraint solver* which attempts to compute a normalized constraint that is equivalent to ϕ, ψ. Note that this requires the underlying constraint language to have a set of normal $\mathcal{L}$-constraints, as does the one presented in the next section. If such a normal constraint cannot be computed, we immediately try another clause since this part of the search space cannot contain any answer. This kind of constraint solving is related to the 'unification' operation in Prolog or PATR-II. Thus we can define the following *optimized goal reduction rule*:

$$p_1, \ldots, p(\vec{x}), \ldots, p_n, \phi \Longrightarrow p_1, \ldots, q_1, \ldots, q_m, \ldots, p_n, \rho$$

where $p(\vec{x})$ is the selected element of a goal and

$$p(\vec{x}) \leftarrow q_1, \ldots, q_m, \psi$$

is a variant of a clause of a definite clause specification S, and ρ is the most general unifier obtained by unifying ϕ and ψ, which we also write as $\rho = \text{UNIFY}(\phi, \psi)$.[2]

A proof of a goal g for a clause specification S is a sequence of goals $G, G_1, \ldots$ where each goal G_{i+1} is derived from G_i by applying goal reduction using a variant of a clause of S and the last goal is the empty clause, where its associated constraint is said to be the *computed S-answer* of the goal g. Höhfeld and Smolka show that answers computed in that way are answers for the goal.

A proof of a goal as described above is a *refutation proof*, which shows that the denial of some formula q is inconsistent with the assumptions of S, i.e., if $S \cup \{\leftarrow q\}$ derives the empty clause. If the empty clause is obtained, then a *refutation* has been discovered, and the constraint ϕ associated with the empty clause is an answer of the goal.

[2]In the remainder of the thesis we will make use of the optimized goal reduction rule, and will use the expressions 'goal reduction' and 'optimized goal reduction' synonymously, unless otherwise specified.

The sequence of goals obtained during a proof is called an *SLD-derivation.* A SLD-derivation may be finite or infinite. A finite SLD-derivation may be successful or failed. A successful derivation is just a refutation. A failed SLD-derivation is one that ends in a non-empty goal with the property that the resulting constraint is unsatisfiable.

The SLD-refutation by itself does not define an algorithm since it does not make use of a concrete *selection function* (or computation rule) as well as a strategy for selecting clauses. The first nondeterminism is known as *don't care* and the second as *don't know.* The don't care property of the selection function is also known as the "independence property" of the selection function (cf. [Lloyd, 1987]) and means that in principle a CLP system can choose any local selection function it finds convenient.

The don't know property means that for the selection of clauses an exhaustive search is necessary, since it cannot be known in advance which sequence of clauses will lead to a successful proof. Since for the selected element of a goal several alternative clauses may be available leading to a set of alternative re-solvents, the search space is a certain type of tree, called an *SLD-tree.* Clearly, the whole search space is only defined implicitly by a definite clause specification, so that an SLD-tree has to be constructed during the proof of some goal. The strategy used to search an SLD-tree is defined by the *search rule.*

An SLD-tree for $S \cup \{g\}$ of a definite clause specification S and a goal g is a tree satisfying the following condition (cf. also [Lloyd, 1987]):

- Each node of the tree is a (possible empty) goal
- The root node is g
- Let $\leftarrow p_1, \ldots, p(\vec{x}), \ldots, p_n, \phi$ be a node in the tree and $p(\vec{x})$ the selected element. Then each possible resolvent (using all matching clauses from S) is a child of that node
- Nodes which are the empty clause have no children.

Each branch of an SLD-tree is a derivation of $S \cup \{g\}$, where those branches which correspond to successful derivations are called *success branches*, and branches corresponding to failed derivations are called *failure branches.* This implies that each success branch corresponds to a computed answer of g.

A *search rule* is a strategy for searching SLD-trees to find success branches. An *SLD-refutation procedure* is specified by a selection function together with a search rule. For example, Prolog is a SLD-refutation procedure using the leftmost selection function and a top-down, depth-first backtracking search rule.

In the next chapter we present a SLD-refutation procedure using a data-driven selection function and an Earley-type search rule. However, in order to illustrate this new algorithm by concrete examples, we first have to define a specific constraint language.

3.2.2 The Constraint Language $\mathcal{L}$

We now present an instance of a constraint language that we are going to use in this thesis. The language is based on the definition of [Smolka, 1992]. Smolka provides us with a very expressive constraint language including feature equation, conjunction, disjunction, negation, and existential quantification. For the purpose of this thesis it suffices to use only a small subset of Smolka's constructions, namely feature equation and conjunction.

Although we only use simple constructions in order to highlight the new results in a clean but simple way, the generalization of Höhfeld and Smolka's scheme guarantees that the results of this thesis also carry over to more complex constraint languages.[3]

The same subset has also been used by [VanNoord, 1993] (because of the same reasons), and following him, we call the "constraint" constraint language $\mathcal{L}$.

Feature Description

We assume three pairwise disjoint sets VAR of variables, C of constants, and L of features.

Given a feature structure, a sequence of labels is used to extract a substructure. Such sequence of features is called a *path* and defined as an expression over L^* (ϵ will be used to indicate the empty path). Constants are viewed as primitive unstructured informational elements.

A *descriptor* is a sequence sp, where s is either a variable or a constant and p is a (possible empty) path.[4] A *feature equation* (or *atomic constraint*) is defined as the equality between descriptors, where $\doteq$ is used as the equality symbol. Thus atomic constraints are of the form

$$d_1 \doteq d_2$$

where d_1 and d_2 are both descriptors. An $\mathcal{L}$-constraint ϕ is an atomic constraint or a conjunction of $\mathcal{L}$-constraints, written as $\phi_1, \ldots, \phi_n$. Note that as more atomic constraints are included, the formula describes fewer feature structures, that is, it becomes less partial and more defined. Thus, these descriptions allow for the structure, partiality and equationality of information [Shieber, 1989]. For example, given that $\{X_1,\ X_2\} \in VAR$, {syn, agr, number, person} $\in L$, and {sg, 3} $\in C$, then

$$\begin{array}{lcl} X_1 \ \mathit{syn\ agr\ number} & \doteq & \mathit{sg}, \\ X_2 \ \mathit{syn\ agr} & \doteq & X_1 \ \mathit{syn\ agr} \end{array}$$

[3]This means that our approach will also work for the whole set of constructions provided by Smolka or for other complex constraint languages (see, e.g., [Backofen and Smolka, 1993; Aït-Kaci *et al.*, 1994]).

[4]Here, we are following the notation given in [VanNoord, 1993].

is an $\mathcal{L}$-constraint denoting some feature structure in which there is a substructure accessible via the path *syn agr* when the value of the feature *number* is constrained to be the constant *sg* and which can be accessed via the *syn* label of two different substructures (denoted by the variables X_1 and X_2). Since, the '*agr*' substructure is part of both 'outer' substructures it is also said that they *share* a substructure. However, for the 'agr' feature it is only required that if a number label is present its value must be *sg*. If we add further atomic constraints to this substructure we are able to express more information. For instance, if we add the atomic constraint

$$X_2 \ syn \ agr \ person \ \doteq \ 3$$

we furthermore require that if the *person* label is present its value must be 3.

Feature Graphs

The semantics of $\mathcal{L}$-constraints will be defined with respect to the domain of feature graphs. A feature graph is a finite, rooted, connected and directed graph for which the following properties must hold :

- Edges are labeled with features.
- For every node, the labels of the out-coming edges must be pairwise distinct.
- Every inner node must be a variable and
- every leaf node must be either a constant or a variable.

For example, figure 3.1 shows two possible dgs for the $\mathcal{L}$-constraint:

$$\begin{array}{lcl} X_1 \ syn \ number & \doteq & sg, \\ X_2 \ syn & \doteq & X_1 \ syn, \\ X_0 \ dtrs \ first & \doteq & X_1, \\ X_0 \ dtrs \ second & \doteq & X_2 \end{array}$$

Formally, a *feature graph* is defined as follows (cf. [Smolka, 1992]) where xfs is called an f-edge (with $x \in VAR$, $f \in L$, and $s \in VAR \cup C$):

- a pair $(c, \emptyset)$, where $c \in C$ and $\emptyset$ is the empty set; or
- a pair (x_0, E), where x_0 is a variable (called the **root**) and E is a finite, possible empty set of edges such that
  1. if xfs and xft are in E, then $s = t$ (i.e., features are *functional*)
  2. if $xfs \in E$, then E contains edges leading from the root x_0 to the node x (i.e., the graph is *connected*)

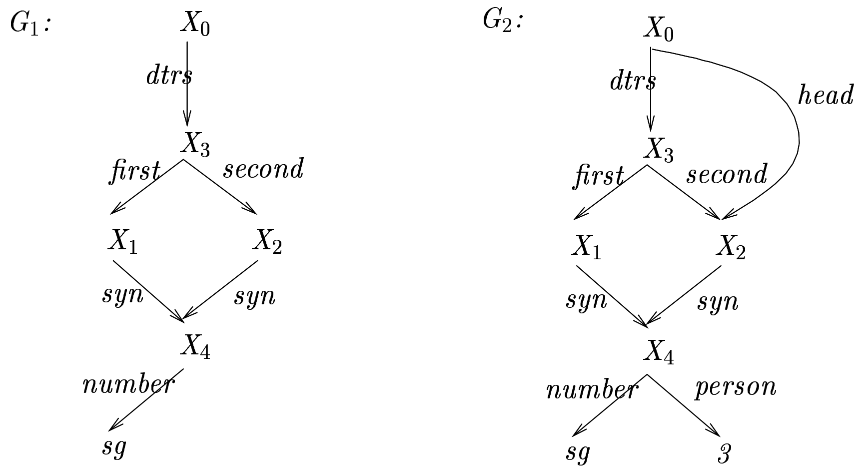
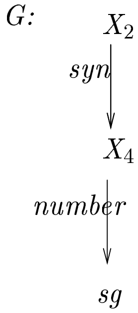


Figure 3.1: The left dg G_1 directly mirrors the set of atomic constraints expressed in the example $\mathcal{L}$-constraint, and the right dg G_2 bears additional constraints. Hence, it is more informative than the left one.

Thus, variables are labels of nodes, features are labels of edges, and constants are the terminal nodes of a feature graph, since no edge can leave a constant node.

A feature graph G is called a *subgraph* of a feature graph G' if the root of G is a variable or a constant occurring in G' and every edge of G is an edge of G'. For example,



is a subgraph of the dg G_1 of figure 3.1. The subgraphs of a feature graph G are partially ordered by

$$G' \leq G'' \iff G' \text{ is a subgraph of } G''$$

For example, for the subgraphs of G_1 of figure 3.1 given in figure 3.2 it holds that $G_3 \leq G_4$, $G_5 \leq G_4$, $G_5 \not\leq G_3$.

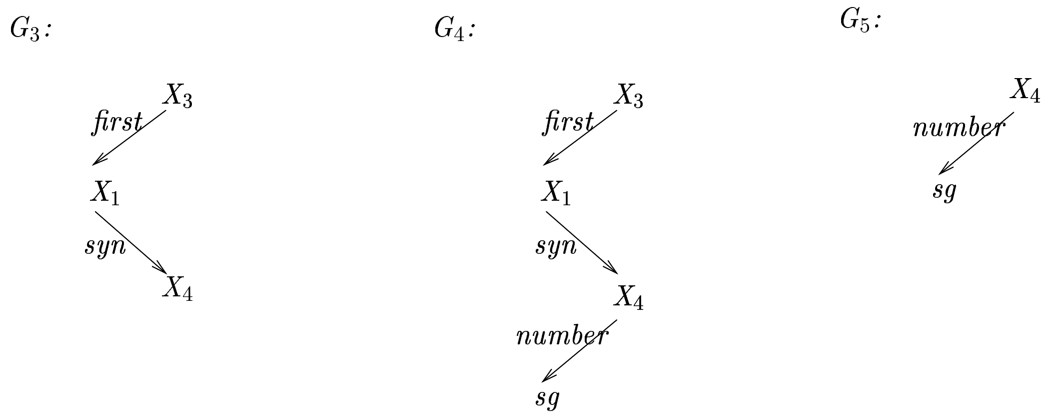


Figure 3.2: Some of the sub-dgs of the dg G_1 given in figure 3.1

According to [VanNoord, 1993] we define the traversal of a given feature graph and a given path as follows: For $p = f_1 \dots f_k$ a path, and G a feature graph, and x a node of G, define x/p to be a node in G as follows: If k=0, then $x/p = x$. Otherwise, $x/p = x^{'}/f_2 \dots f_k$, if there exists an f_1-edge $x f_1 x'$ (otherwise, x/p is undefined). We will use the notation G/p to mean x_0/p for x_0 the root node of G.

If G is a feature graph and s is a constant or variable in G, then G_s denotes the unique maximal subgraph of G whose root is s. For a feature graph G and a path p, G_p denotes the subgraph G_s, if $G/p = s$ is defined.

For example the subgraph G_{X_2} of the graph G_1 of figure 3.1 (i.e., the unique maximal subgraph of G_1 with root node X_2) is denoted by the expression $G_{1_{dtrs\ second}}$.

Interpretation of Constraints

An interpretation $\mathcal{I}$ of $\mathcal{L}$ consists of a domain $\mathcal{D}^{\mathcal{I}}$ which is the set of all feature graphs, and an interpretation function $[\![.]\!]^{\mathcal{I}}$ (see definition 1). Variables and constants will denote feature graphs, relative to some assignment.

The denotation of a variable X with respect to an assignment α is simply $\alpha(X)$. The denotation of a constant C is the feature graph $(c, \emptyset)$ (for any assignment). The denotation of a descriptor *sp* is the subgraph at path p of the graph denoted by s.

An interpretation $\mathcal{I}$ *satisfies* an atomic constraint $d_1 \doteq d_2$ relative to an assignment α, if the descriptors are both defined and the same, i.e.,

$$\mathcal{I} \models_\alpha d_1 \doteq d_2 \text{ iff } [\![d_1]\!]^{\mathcal{I}}_\alpha = [\![d_2]\!]^{\mathcal{I}}_\alpha$$

A constraint is called satisfiable if it has a solution, and two constraints ϕ and ψ are called equivalent if they have the same solutions, i.e., if $[\![\phi]\!]^{\mathcal{I}} = [\![\psi]\!]^{\mathcal{I}}$. Clearly, not all constraints are satisfiable. For example, the constraint $c_1 \doteq c_2$ is not satisfiable, since both denote different graphs for all assignments.

The problem of whether a constraint is satisfiable is decidable. For example in [Smolka, 1992] an algorithm for a more powerful feature logic is presented. In [VanNoord, 1993] a decidable algorithm for the constraint language $\mathcal{L}$ is presented, which we briefly summarize in the next paragraph.

Satisfiability of a constraint ϕ is shown by transforming ϕ into a constraint of a specific form, called the *normal form*. This transformation is performed in two steps. Firstly, all complex paths (paths containing more than one label) are removed by introduction of some new variables. The resulting constraint (also called *basic*) is shown to be satisfiable iff the original constraint was. The next step then rewrites constraints without complex path expressions into *normal form*.

If the resulting normal constraint is *clash free*, i.e., if it does not contain any constraints of the following form:

- $cl \doteq d$ (constant/compound clash)
- $c_1 \doteq c_2$ (constant clash)

then it is called a *solved* clause. A solved clause C is of the form $Xl \doteq s$ or $X \doteq s$.

Readable Notation We are using a special representation for complex constraints, called matrix notation (see also [VanNoord, 1993]). In order to distinguish variable names from features and constants we adopt the Prolog convention that only variable names start with a capitalized letter and names of features or constants have to be written in lower case. For example the following constraints on the variable X_0

$X_0\ f_1\ f_3 \doteq$ c ,
$X_0\ f_2 \quad \doteq X_0\ f_1\ f_3$

are represented in matrix notation as follows (the variables X_1 and X_2 are computed during the computation of the basic constraint):

(1) $$_{X_0}\begin{bmatrix} f_1 & {}_{X_1,X_2}\begin{bmatrix} f_3 & c \end{bmatrix} \\ f_2 & c \end{bmatrix}$$

If variables occur only once in a matrix they are omitted. This is the case for the above example, so that it will also be written as:

(2) $$\begin{bmatrix} f_1 & \begin{bmatrix} f_3 & c \end{bmatrix} \\ f_2 & c \end{bmatrix}$$

Furthermore, empty feature structures will not been shown explicitly, i.e.,

(3) $${}_{X_0}\begin{bmatrix} f & {}_{X_1}[\ ] \end{bmatrix}$$

is written as

(4) $_{X_0}\begin{bmatrix} f\ X_1 \end{bmatrix}$

For feature structures encoding lists I will adopt the list notation from HPSG[Pollard and Sag, 1987]. The feature structure encoding of the following list

$$(5)\quad \begin{bmatrix} \texttt{first}\ a \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ b \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ c \\ \texttt{rest}\ \ end \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

will be written more readable using angled brackets

(6) $\langle a\ b\ c\rangle$

The empty list then will be written as

(7) $\langle\ \rangle$

We will also make use of the head/tail representation of lists known from Prolog. Thus to explicitly represent the first element of a list from the rest we write

(8) $\langle First|Rest\rangle$

thus, for instance, $\langle a,\ b,\ c\rangle$ can also be written as $\langle a| <\ b,\ c >\rangle$.

The difference list of the feature structure

$$(9)\quad \begin{bmatrix} \texttt{dl}\ \begin{bmatrix} \texttt{first}\ a \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ b \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ c \\ \texttt{rest}\ \ {}_{X_1}[\ ] \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \texttt{el}\ \ {}_{X_1}[\ ] \end{bmatrix}$$

will be written as

(10) $\langle a\ b\ c|X_1\rangle$-$X_1$

The empty difference list will be written as

(11) X-X

3.3 Specification of Constraint-based Grammars in $\mathcal{R}(\mathcal{L})$

Using the formalism of [Höhfeld and Smolka, 1988] a constraint-based grammar would formally be considered as a definite clause specification using some specific constraint language. In their simplest form parsing and generation are then viewed as proof procedures that try to find answers for a given goal with respect to a given grammar.

Unless we specify what a definite clause specification intended to represent a natural language grammar, linguistically means, there is obviously no distinction between parsing and generation possible, and computational linguistics would just be constraint-logic programming. In the same sense as conventional programming languages are used just as computational means to specify algorithmic solutions of some problem domain, we adopt the view that CLP is just a formal and operational tool to specify grammatical theories. The theories itself have to give criteria for giving parsing and generation a different meaning which they intuitively seem to convey. It is then a matter of effectivity and efficiency whether they can be realized by the same process or whether two specialized processes have to be developed.

Clearly, those people who are interested in computational aspects of natural language cannot wait until the one and only grammatical theory has been found because grammar theory development certainly undergoes (scientific) evolution. The basic advantage of currently developed theories that follow the constraint-based view, however, is that they present an important homogeneous view with respect to the level of information they try to model that also fits very well with current developments in CLP. By taking into account this commonality when developing parsing and generation strategies it is possible to obtain efficient computability for a broad class of linguistic theories.

Although we do not want to make too many restrictions on the specific form of a constraint-based grammar we want to process, we have to make some assumptions about the representation of phonological and semantic information, and how linguistic objects can be combined to form larger objects.

Form of grammar rules and lexical entries A grammar G is specified as a definite clause specification where the literals of each definite clause are unary relational atoms. Considering only unary atoms is not a general restriction since by means of reification (see [Pereira and Shieber, 1987], [Genesereth and Nilsson, 1987]) we can also express an n-ary atom $\mathrm{r}(\vec{X})$ in terms of constraints of a unary relation s(Y) using for example the features REL and ARG_i such that the relational symbol r is viewed as a constant bound to the feature REL and each variable x_i is bound to the corresponding feature ARG_i. Thus $\mathrm{r}(\vec{X})$ would be represented as follows

$$s(Y), Y\ rel\ \doteq\ r, Y arg_i\ \doteq\ X_i$$

Thus the general form of a grammar rule is as follows:

$$p(x_0) \leftarrow q_1(x_1) \ldots q_n(x_n)$$

The relational atoms are assumed to denote possible constituents of a grammar, either specifically (using for each possible constituent a specific symbol, like NP, VP, PP) or schematically by only using one symbol, e.g., SIGN. For example, the rule (i.e., the definite clause)

$$vp \leftarrow v,\ np,\ pp,\ \phi$$

expresses that a verb phrase VP consists of a verb V, a nominal phrase NP and of a prepositional phrase PP and the following rule

$$sign \leftarrow sign,\ sign,\ \phi$$

expresses that a phrase is built from two phrases, no matter what they are (as long as we do not consider the constraint ϕ). Although the last rule seems to be useless, since it does not say very much about the actual structure of an object, this kind of schematic rule is very prominent in *lexicalized grammars*, since they allow the specification of general combinatory rules, which are independent from individual words (see [Uszkoreit, 1986b] for more details of such lexicalized view). In fact, the grammar that we are going to use in this thesis and which can be found in appendix A belongs to this kind of grammars.

Using this notation, we will define lexical entries as unit clauses, and grammar rules as non-unit clauses (defining non empty productions) as well as unit clauses (defining empty productions). In order to distinguish between lexical entries and empty productions we will use the boolean feature LEX.

Representation of phonological information In the case of phonological information of an utterance we make the simplified assumption that it is represented as a string, i.e., we represent the phonological structure of a sentence as a list of words. At least in the case of written text this simplification is not that critical and because most of todays natural language systems are designed for processing written text we are in good company (but see for example [ICSLP, 1992]).

We adapt the difference list notation known from standard Prolog interpreters for Definite Clause Grammars. Thus the utterance "Peter loves Mary." will be represented as (using the matrix notation):

$$\begin{bmatrix} \texttt{phon} & \begin{bmatrix} \texttt{dl} & \begin{bmatrix} \texttt{first} & \textit{peter} \\ \texttt{rest} & \begin{bmatrix} \texttt{first} & \textit{loves} \\ \texttt{rest} & \begin{bmatrix} \texttt{first} & \textit{mary} \\ \texttt{rest} & *\textit{end} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \texttt{el} & *\textit{end} \end{bmatrix} \end{bmatrix}$$

or more readable (using the notation introduced above)

$$\begin{bmatrix} \texttt{phon} & \langle peter, loves, mary \rangle \text{-} \langle\ \rangle \end{bmatrix}$$

The phonological information of a lexical entry like "peter" will be of form

$$\begin{bmatrix} \texttt{phon} & \langle peter | T \rangle \text{-} T \end{bmatrix}$$

Now it is very easy to represent a context-free grammar as a definite clause specification in $\mathcal{R}(\mathcal{L})$. For example for the simple context-free grammar

$$\begin{array}{lcl} \text{s} & \rightarrow & \text{np vp} \\ \text{np} & \rightarrow & \text{peter} \\ \text{vp} & \rightarrow & \text{sleeps} \end{array}$$

a possible $\mathcal{R}(\mathcal{L})$ grammar is

$$\begin{array}{lcl} \text{s(X)} & \leftarrow & \text{np(Y),vp(Z),} \\ & & \text{X phon dl} \doteq \text{Y phon dl,} \\ & & \text{Y phon el} \doteq \text{Z phon dl,} \\ & & \text{X phon el} \doteq \text{Z phon el} \\ \text{np(X)} & \leftarrow & \text{X phon dl first} \doteq \text{peter,} \\ & & \text{X phon dl rest} \doteq \text{X phon el} \\ \text{vp(X)} & \leftarrow & \text{X phon dl first} \doteq \text{sleeps,} \\ & & \text{X phon dl rest} \doteq \text{X phon el} \end{array}$$

Using $\mathcal{R}(\mathcal{L})$ constraints we have explicitly to specify the concatenation of substrings to build larger strings, implicitly specified in context-free grammar rules. However, using the difference list notation this is easily realized.

Note that since we have to explicitly specify how strings are combined to build larger strings, it is also possible to specify string combinations other than by simple concatenation, see for example [Gerdemann, 1991; VanNoord, 1993]. To illustrate this we adopt an example of a categorial style grammar from [Gerdemann, 1991].

$$\begin{array}{lcl} \text{sign(X0)} & \leftarrow & \text{sign(X1),sign(X2),} \\ & & \text{X0 phon} \doteq \text{X1 phon,} \\ & & \text{X1 arg} \doteq \text{X2} \\ \text{sign(X)} & \leftarrow & \text{X phon dl first} \doteq \text{tom,} \\ & & \text{X phon dl rest} \doteq \text{X phon el} \\ \text{sign(X)} & \leftarrow & \text{X phon dl first} \doteq \text{friends,} \\ & & \text{X phon dl rest} \doteq \text{X phon el} \\ \text{sign(X)} & \leftarrow & \text{X phon dl first} \doteq \text{call,} \\ & & \text{X phon dl rest} \doteq \text{X arg phon dl,} \\ & & \text{X phon el} \doteq \text{X arg phon el rest,} \\ & & \text{X arg phon el first} \doteq \text{up} \end{array}$$

which can be represented more readable as follows:

$$\mathrm{sign}(\begin{bmatrix}\mathtt{phon:} & X0\end{bmatrix}) \quad \leftarrow \quad \mathrm{sign}(\begin{bmatrix}\mathtt{phon:} & X0\\ \mathtt{arg:} & X1\end{bmatrix}),\mathrm{sign}(\mathrm{X1})$$

$$\mathrm{sign}(\begin{bmatrix}\mathtt{phon:} & \langle Tom|T\rangle\text{-}T\end{bmatrix})$$

$$\mathrm{sign}(\begin{bmatrix}\mathtt{phon:} & \langle Friends|T\rangle\text{-}T\end{bmatrix})$$

$$\mathrm{sign}(\begin{bmatrix}\mathtt{phon:} & \langle call|R\rangle\text{-}T\\ \mathtt{arg:} & \begin{bmatrix}\mathtt{phon:} & \langle R\rangle\text{-}\langle up|T\rangle\end{bmatrix}\end{bmatrix})$$

The last entry is an example of the *head wrapping* functor, *call up*, which shows that this approach can accommodate operations other than simple concatenation.

Representation of semantic information Recently, a constraint-based view of semantic representation has become quite popular in the area of computational semantics, e.g., [Fenstad *et al.*, 1987], [Pollard and Sag, 1987], [Alshawi and Pulman, 1992], and [Nerbonne, 1992]. The main advantage of representing semantic information as feature structures is that it allows to express a simple and systemic syntax/semantic interface, since "... it harmonizes so well with the way in which syntax is now normally described; this close harmony means that syntactic and semantic processing ... can be tightly coupled as one wishes – indeed, there needn't be any fundamental distinction between them at all. In feature-based formalisms, the structure shared among syntactic and semantic values constitutes the interface in the only sense in which this exists." [Nerbonne, 1992], page 3. Thus the constraint-based view sees the interface as being specified as a set of constraints, to which non-syntactic information (e.g., phonological or even pragmatic information) may contribute.

From a processing point of view, the advantages of viewing semantic information directly as part of a constraint-based grammar, is that not only a parallel view on the different levels of description is possible, but that the relationship between these levels can be stated completely declaratively. Processing of semantic information can then be performed in tandem with the processing of syntactic information, using the same basic constraint-solving mechanism, e.g., unification. This means that there are no special processes needed for mapping syntactic information to semantic information and vice versa (at least with respect to grammatical processing). This is especially useful in the case of generation, where the basic task is to find for given semantic information represented as a feature structure the strings licensed by the grammar.

Although the techniques for processing of reversible grammars are supposed to abstract away from the different particularities of phonological and semantic representation, we have to define some simple semantic structures to be able to illustrate our methods to be developed in the next chapters by some concrete examples. For this reason we will simply represent semantic structures essentially as predicate argument structures (following [VanNoord, 1993]). For example, the binary predicate 'erzählen' (meaning 'to tell') will be represented as follows:

$$\begin{bmatrix} \texttt{sort} \ \textit{binary} \\ \texttt{pred} \ \textit{erzählen} \\ \texttt{arg1} \ X \\ \texttt{arg2} \ Y \end{bmatrix}$$

where the feature PRED specifies the name of the predicate, the value of SORT specifies the arity, and the features ARG1 and ARG2 hold the semantic structures of the arguments. As another example consider the representation of the null-ary predicate 'lügen' (meaning of the noun 'lies'):

$$\begin{bmatrix} \texttt{sort} \ \textit{nullary} \\ \texttt{pred} \ \textit{lügen} \end{bmatrix}$$

If we assume that semantic structures are bound to the feature SEM then the simplified relationship between the phonological string "peter erzählt lügen" and its semantic representation 'erzählen(peter,lügen)' would be the feature structure

$$\begin{bmatrix} \texttt{phon} & \langle \textit{peter}, \textit{erzählt}, \textit{lügen} \rangle \text{-} \langle\ \rangle \\ \texttt{sem} & \begin{bmatrix} \texttt{sort} \ \textit{binary} \\ \texttt{pred} \ \textit{erzählen} \\ \texttt{arg1} \begin{bmatrix} \texttt{sort} \ \textit{nullary} \\ \texttt{pred} \ \textit{peter} \end{bmatrix} \\ \texttt{arg2} \begin{bmatrix} \texttt{sort} \ \textit{nullary} \\ \texttt{pred} \ \textit{lügen} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Modifier constructions such as noun-adjective constructions or adverbial modifications will be represented using the feature MOD, that holds (the possibly complex) semantic structure of the modifier. However, instead of placing the MOD feature at the same level as the ARGs feature we will bundle the semantics of the modified predicate argument structure under the feature ARG1. Thus a modifier construction consists of a feature structure with top-level feature MOD and ARG1. The sortal value of such constructions will be restricted to the value MODIFIER. Therefore, the semantic structure of an utterance "Heute erzählt peter gerne lügen" may look as follows:

$$
\left[\texttt{sem}\ \left[\begin{array}{ll}
\texttt{sort} & \textit{modifier} \\
\texttt{mod} & \left[\begin{array}{ll}\texttt{sort} & \textit{nullary} \\ \texttt{pred} & \textit{heute}\end{array}\right] \\
\texttt{arg1} & \left[\begin{array}{ll}
\texttt{sort} & \textit{modifier} \\
\texttt{mod} & \left[\begin{array}{ll}\texttt{sort} & \textit{nullary} \\ \texttt{pred} & \textit{gerne}\end{array}\right] \\
\texttt{arg1} & \left[\begin{array}{ll}
\texttt{sort} & \textit{binary} \\
\texttt{pred} & \textit{erzählen} \\
\texttt{arg1} & \left[\begin{array}{ll}\texttt{sort} & \textit{nullary} \\ \texttt{pred} & \textit{peter}\end{array}\right] \\
\texttt{arg2} & \left[\begin{array}{ll}\texttt{sort} & \textit{nullary} \\ \texttt{pred} & \textit{lügen}\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]\right]
$$

Representation of grammatical derivations We are also interested in the derivational history of a parsed or generated expression, i.e., in the derivation tree which represents how a certain derivation is licensed 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.

We will represent such a derivation tree as a feature structure value of the feature DERIV. Each head of a definite clause of the grammar (i.e., the grammar rules and lexical entries) has to have an additional feature LABEL which value is a constant that uniquely identifies this clause. The feature DTRS is used to express the relationship between the DERIV features of the body of a clause and its head. For lexical entries and for empty productions, the value of this DTRS feature is the empty list, and for grammar rules other than empty productions, the value will be a list whose sequence corresponds with that of the elements of the body. Thus the general structure of a definite clause with the DERIV feature is (using our abbreviations for feature structures):

$$
h(\left[\textsc{deriv}\ \left[\begin{array}{ll}\textsc{label} & \textit{“name”} \\ \textsc{dtrs} & \langle \textit{deriv}_1 \ldots \textit{deriv}_n \rangle\end{array}\right]\right]) \leftarrow b_1(\left[\textsc{deriv}\ \textit{deriv}_1\right]), \ldots, b_n(\left[\textsc{deriv}\ \textit{deriv}_n\right])
$$

For unit clauses (representing for example lexical entries) we have the general form

$$
h(\left[\textsc{deriv}\ \left[\begin{array}{ll}\textsc{label} & \textit{“name”} \\ \textsc{dtrs} & \langle\ \rangle\end{array}\right]\right])
$$

Using the simple context-free grammar we can adapt it for representing the derivation feature as follows:

$$
\begin{array}{lll}
s(X) & \leftarrow & np(Y), vp(Z), \\
 & & X \text{ phon dl} \doteq Y \text{ phon dl}, \\
 & & Y \text{ phon el} \doteq Z \text{ phon dl}, \\
 & & X \text{ phon el} \doteq Z \text{ phon el}, \\
 & & X \text{ DERIV LABEL} \doteq s1, \\
 & & X \text{ DERIV DTRS first} \doteq Y \text{ deriv}, \\
 & & X \text{ DERIV DTRS rest first} \doteq Z \text{ deriv}, \\
 & & X \text{ DERIV DTRS rest rest} \doteq {}^*\text{end} \\
np(X) & \leftarrow & X \text{ phon dl first} \doteq \text{peter}, \\
 & & X \text{ phon dl rest} \doteq X \text{ phon el}, \\
 & & X \text{ DERIV LABEL} \doteq \text{np-peter}, \\
 & & X \text{ DERIV DTRS} \doteq {}^*\text{end} \\
vp(X) & \leftarrow & X \text{ phon dl first} \doteq \text{sleeps}, \\
 & & X \text{ phon dl rest} \doteq X \text{ phon el}, \\
 & & X \text{ DERIV LABEL} \doteq \text{vp-sleeps}, \\
 & & X \text{ DERIV DTRS} \doteq {}^*\text{end}
\end{array}
$$

For the string "Peter sleeps" the resulting derivation tree represented as a feature structure is the following:

$$
\left[\text{DERIV} \begin{bmatrix} \text{LABEL} & \textit{s1} \\ \text{DTRS} & \left\langle \begin{bmatrix} \text{LABEL} & \textit{np-peter} \\ \text{DTRS} & \langle\ \rangle \end{bmatrix}, \begin{bmatrix} \text{LABEL} & \textit{vp-sleeps} \\ \text{DTRS} & \langle\ \rangle \end{bmatrix} \right\rangle \end{bmatrix}\right]
$$

Note that the sequence of the daughters derivation directly reflects the sequence of elements of the clause's body. However, it does not indicate in which order the elements of the body have been processed. Furthermore, since we only require that the rule name identifier should be present in the derivation tree representation, the derivation tree represented in the grammar only specifies "a backbone", i.e., it only says, which rules have been applied. However, applying these rules exactly in this order (or structure) would "replay" the successful derivation of the result, which finally would result in the same feature structure.

We say that the value of a DERIV feature is *complete*, if every element of the DTRS feature is complete. The DERIV value of a unit clause is complete by definition, because its DTRS feature is empty. This means a value of the DERIV feature of a found answer is complete.

We say that the value of a DERIV feature is *partial*, if there exists an element of the DTRS feature, which is not complete. Since, this incomplete daughter element corresponds to an element of the body of a clause, this means that for this clause not all elements have been processed.

Readable notation In the following chapters, we make use of a more readable and simplified representation for $\mathcal{R}(\mathcal{L})$-constraints. Using the matrix notation introduced for representing $\mathcal{L}$-constraints, we will often leave off the constraint in a definite clause, and instead replace the variables in the clause with the matrix notation of the constraint on those variables.

Thus instead of

$$
\begin{array}{l}
\mathit{sign}(X_0) \leftarrow \\
\quad \mathit{sign}(X_1), \\
\quad \mathit{sign}(X_2), \\
\quad X_0 \ \mathit{syn}\ \mathit{cat} \doteq s, \\
\quad X_1 \ \mathit{syn}\ \mathit{cat} \doteq \mathit{np}, \\
\quad X_2 \ \mathit{syn}\ \mathit{cat} \doteq \mathit{vp}, \\
\quad X_1 \ \mathit{syn}\ \mathit{agr} \doteq X_2 \ \mathit{syn}\ \mathit{agr}
\end{array}
$$

I write:

$$
\begin{array}{l}
\mathit{sign}(\ {}_{X_0}\begin{bmatrix}\texttt{syn} \begin{bmatrix}\texttt{cat}\ s\end{bmatrix}\end{bmatrix}\ ) \leftarrow \\
\quad \mathit{sign}(\ {}_{X_1}\begin{bmatrix}\texttt{syn} \begin{bmatrix}\texttt{cat}\ \mathit{np} \\ \texttt{agr}\ \mathit{Agr}\end{bmatrix}\end{bmatrix}\ ), \\
\quad \mathit{sign}(\ {}_{X_2}\begin{bmatrix}\texttt{syn} \begin{bmatrix}\texttt{cat}\ \mathit{vp} \\ \texttt{agr}\ \mathit{Agr}\end{bmatrix}\end{bmatrix}\ )
\end{array}
$$

In the case where variables occur only once I will omit them. Thus the above clause can also be written as:

$$
\begin{array}{l}
\mathit{sign}(\begin{bmatrix}\texttt{syn} \begin{bmatrix}\texttt{cat}\ s\end{bmatrix}\end{bmatrix}\ ) \leftarrow \\
\quad \mathit{sign}(\begin{bmatrix}\texttt{syn} \begin{bmatrix}\texttt{cat}\ \mathit{np} \\ \texttt{agr}\ \mathit{Agr}\end{bmatrix}\end{bmatrix}\ ), \\
\quad \mathit{sign}(\begin{bmatrix}\texttt{syn} \begin{bmatrix}\texttt{cat}\ \mathit{vp} \\ \texttt{agr}\ \mathit{Agr}\end{bmatrix}\end{bmatrix}\ )
\end{array}
$$

3.4 Parsing and Generation

From the point of view of parsing and generation, a grammar as considered above defines a relation between strings of a natural language and representations of the meaning modelled as part of the grammar, which we call semantic expressions. Parsing then involves the computation of this relation from strings to semantic expressions, and generation involves the computation from semantic expressions to strings.

More formally this relationship can be defined as a binary relation R between objects of two different domains, i.e., $R \subseteq S \times LF$, where S is the domain of strings and LF the domain of semantic expressions.

Parsing as well as generation can be thought of as a program P that is able to enumerate all possible pairs of R for a given element either from the domain of strings or from the domain of semantic expressions.[5] More precisely, in the case of parsing P computes $\{lf_i | < s, lf_i > \in R\}$ and in the case of generation $\{s_i | < s_i, lf > \in R\}$. Thus, P is just a constructive realization of R, no matter whether P constructs R only during parsing or during generation.

Since P can construct R for both domains we call P a *reversible* program and R a *P-reversible* relation, in order to emphasize that P can construct R from both directions.

Clearly, up to now we have only assumed that R is a (recursively) enumerable relation. As usual, we assume that the set S of the well-formed strings of a language is enumerable. For a reversible program P this implies that it can enumerate R also from the set LF. Furthermore, we also assume that at least S has an infinite cardinality, so that R has to be defined by some finite recursive device, i.e., a grammar. Intuitively we assume that the same grammar is used for defining both sets of R, therefore we will call this grammar a *reversible grammar*.

We have only assumed that P should be able to compute a reversible relation. In principle this does not exclude the case that there are infinitely many solutions for some $s \in S$ or $lf \in LF$. For example this would imply that there are infinitely many readings for some sentence or infinitely many paraphrases for some logical form. If this is the case then either the grammar is intrinsically infinitely enumerable or P does not terminate (see also [Dymetman, 1991]). Therefore we restrict a program P to be *effectively reversible* in the following sense (see also [VanNoord, 1993]):

- A reversible program P is **effectively reversible** iff
  - P enumerates a relation R
  - P is guaranteed to terminate
- A relation R is *effectively reversible*, iff it can be constructed by an effectively reversible program P. We will say that a grammar is *effectively reversible* iff it defines an effectively reversible relation.

This means that P is decidable and that the enumeration of the set of solutions

[5] Without loss of generality we can assume that parsing and generation are performed by the same program P. In case we assume a specific parser or generator P would simply trigger these algorithms by means of a flag.

must be finite, i.e., for each input (either sentence or logical form) the resulting set is 'one to finitely many'.

Considered under the CLP view, the parsing and generation problem consists of a goal that has to be resolved with respect to a given grammar G, specified as a definite clause specification. Parsing and generation differ with respect to the constraints specified for the goal. Since for parsing we want to find the corresponding semantic expressions to a particular string, we require that the constraints at least entail the representation of the string in question, and analogously for generation we require that the semantic expression for which possible strings should be computed is specified. For parsing the feature that represents the string can be considered as an input variable and the feature that represents the semantics found can be considered as the output variable, and vice versa for generation. We will call the feature that represents the input the *essential feature*, short Ea. For parsing we will assume that Ea is the path PHON and for generation it is SEM.

A parsing goal then can be defined as a goal of which the essential feature is PHON and whose value is bound to the string in question.

Thus, the parsing problem for the string "heute erzählt peter lügen" would be

$$sign(\begin{bmatrix}\texttt{phon} & \langle heute, erzählt, peter, lügen\rangle\text{-}\langle\ \rangle\end{bmatrix})$$

and analogously we define a generation goal as a goal of which the essential feature is SEM and whose value is bound to the semantic expression in question. For example for the logical form "heute(erzählen(peter,lügen))" would be

$$sign(\begin{bmatrix}\texttt{sem} & \begin{bmatrix}\texttt{mod} & heute \\ \texttt{arg1} & \begin{bmatrix}\texttt{pred} & erzählen \\ \texttt{arg1} & peter \\ \texttt{arg2} & lügen\end{bmatrix}\end{bmatrix}\end{bmatrix})$$

Note that in both cases further constraints may be added to restrict the possible feature structures of found results, for example to be of a specific category, or that the subcategorization list should be empty. Moreover, it would also be possible to specify the entire syntactic information, for example in the case of generation, to perform some grammar checking. However, what we at least require for parsing and generation is that the value of the essential feature is instantiated. In the next paragraph we define more precisely what "instantiation of the essential feature means."

Restricted parsing problem So far, we only have required that the value of the essential feature should be instantiated. However, it has not been specified what exactly this means. Informally, it can be specified like "show me all signs that place exactly the following constraints on the following semantic representation."

([VanNoord, 1993], page 56) or string. That is, we want our algorithm to enumerate all possible feature structures that have a *compatible* value for the value of essential feature. Thus if we want to parse a string, we want the feature structure of that string and analogously for generation we want a feature structure of the input semantics. In the case of parsing, this restriction is implicitly obtained, if the string is represented as a proper list and contains no variables.

In [Wedekind, 1988] a formalization of such a criterion has been given for generation, under the term of *coherence* and *completeness*. Let Sem_ϕ be the semantic expression of the goal constraint ϕ and Sem_ψ the semantic expression of the answer constraint ψ. Then a generator is said to be coherent if Sem_ψ subsumes Sem_ϕ, and complete if Sem_ϕ subsumes Sem_ψ. In other words, a generator is said to be complete if all information of the goal semantic expression is considered and it is coherent, if the generator does not add additional semantic information during processing.

Van Noord has generalized this notation under the term *p-parsing problem*, where parsing in this sense is the general notation for parsing of a string and generation of a semantic expression. He gives the following definition. Let the *restriction* of a constraint ϕ with respect to a path p, written as ϕ/p defined as follows:

$$[\![(\phi/p)]\!]^{\mathcal{I}} := \{\beta \in \mathcal{ASS}^{\mathcal{I}} | \exists\alpha \in [\![\phi]\!]^{\mathcal{I}} \mathit{such\ that}\ [\![p]\!]^{\mathcal{I}}_{\alpha} = [\![p]\!]^{\mathcal{I}}_{\beta}\}$$

Then the p-parsing problem consists of a grammar G and a goal q such that

$$\leftarrow q(X), \phi$$

A answer to a p-parsing problem is a solved constraint ψ such that

- ψ is an answer q with respect to G; and
- $[\![(\phi/Xp)]\!]^{\mathcal{I}} = [\![(\psi/Xp)]\!]^{\mathcal{I}}$

In our terminology the path p corresponds to the essential feature Ea. Thus we also use the term *Ea-proof problem* to indicate that parsing and generation are proofs of goals in which the value of the essential feature is instantiated.

Ambiguity vs. Paraphrasing The above characterizations express only the case that for parsing and generation the same relation has to be defined by some grammar G. But this does not exclude the possibility that G is compiled into specific parsing and generation grammars, as long as they define the same relation R (providing the compilation step is correct, e.g., [Strzalkowski, 1989; Block, 1991; Dymetman *et al.*, 1990]).

In order to distinguish the two cases, where a reversible grammar G is used only during compile–time or is used during run–time for performing parsing and generation the terms *weakly reversible* grammars and *strongly reversible* grammars are introduced. We will say, that a reversible grammar G is strongly reversible iff P enumerates the respective sets using G during run–time, otherwise G is weakly reversible. In this thesis, we are only interested in strongly reversible grammars.

If a sentence s has been associated with more than one interpretation, say $lf_1 \ldots lf_n$, the relation R defined by G will contain pairs $\langle s, lf_1\rangle \ldots \langle s, lf_n\rangle$ and analogously for a meaning representation lf we will get a set of pairs $\langle s_1, lf\rangle \ldots \langle s_m, lf\rangle$, of all possible sentences that have the same interpretation. Accordingly, the sets are denoted as $R(s)$ or $R(lf)$. The cardinality $\mathrm{card}(R(s))$ of $R(s)$ is defined as the *degree of ambiguity* of s and the cardinality $\mathrm{card}(R(lf))$ of $R(lf)$ as the *degree of paraphrases* of lf.

Suppose that for some s there exists exactly one semantic expression lf, i.e., $\mathrm{card}(R(s)) = 1$. Then, it is not valid to deduce that if generation is performed starting with lf the resulting set $R(lf)$ is $\{s\}$. However, it is guaranteed that $s \in R(lf)$ (see also figure 3.3).

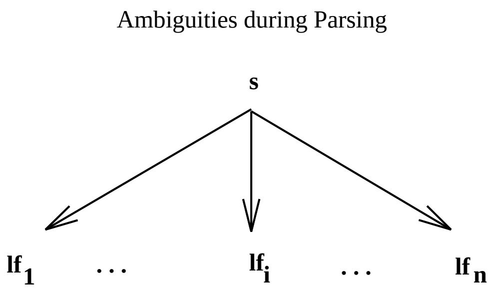


Figure 3.3: The relationship between paraphrases and ambiguities.

Of course, this kind of "reversibility" is an intrinsic property of each relation. But, if two separate grammars for parsing and generation are used in a natural language system it has to be proven that they describe the same relation[6]; otherwise it would be possible that a sentence which is parse-able cannot be generated and vice versa. Grammar reversibility is very important in practice because it ensures that ambiguous structures and its paraphrases are interrelated. If this is not the case then important aspects of performance like self–monitoring or generation of paraphrases in order to disambiguate ambiguous sentences cannot be modelled properly (in chapter 5 we discuss this problem in more detail).

Thus viewed, understanding and generation are *dual* processes, in the sense that each sentence which can be understood should also be producible and vice versa. This kind of duality is naturally captured if reversible grammars are used.

[6]This is also the case if the two grammars are automatically compiled from one competence grammar, see,e.g., [Strzalkowski, 1989; Block, 1991].

3.5 Conclusion

In this chapter we have introduced constraint-based grammars as an appropriate means for specifying reversible grammars. We have introduced the constraint logic programming scheme of [Höhfeld and Smolka, 1988] as an appropriate formal means for representing constraint-based grammars, which also provides an operational semantics in the form of generalized SLD-resolution. Following the constraint-based grammar formalism introduced in [VanNoord, 1993] we have shown how to specify constraint-based grammars by basically specifying the manner of representing phonological and semantic information.

Although we have chosen a simple constraint language in order to highlight the new results in a clean but simple way, the generalization of Höhfeld and Smolka's scheme will guarantee that the results of this thesis also carries over for more complex constraint languages.

At this place we want to emphasize, that we had consciously used well-known and accepted formalisms rather than defining our own formalism. The main reason is that we are interested in the *application* of constraint-based formalisms for natural language processing. Thus our focus of attention is algorithmic rather than theoretical. However, basing it on accepted theoretical approaches makes our project an attractive and worthwhile venture. We start doing this by presenting an efficient uniform tabular algorithm for parsing and generation of constraint-based grammars.

Chapter 4

A Uniform Tabular Algorithm for Parsing and Generation

In this chapter a practical uniform algorithm for processing of reversible natural language grammars is presented that can be used for efficient parsing and generation of constraint-based grammars without the need of compilation. Hence, we call this approach an *interpreter for reversible grammars*.

The new approach follows the paradigm of "natural language processing as deduction" as introduced by [Pereira and Warren, 1983] for the case of parsing and extended by [Shieber, 1988] to the case of generation. As described in Shieber's work parsing as well as generation can be thought of as the constructive proving of a string's grammaticality or of the existence of a string that matches some given criterion (e.g., a given semantic expression). Under this deductive view the difference between parsing and generation rests in which information is given as premises and what the goal is to be proved. Thus "parsing and generation can be viewed as two processes engaged in by a single parameterized theorem prover for the logical interpretation of the formalism" ([Shieber, 1988], page 614). This view is the starting point of the new approach developed in this thesis.

In order to model parsing and generation using the same basic control logic in a task specific manner, we will develop the new uniform tabular algorithm along the following dimensions:

- data-driven selection function
- uniform indexing technique
- item sharing

The first basic idea is to use the same set of inference rules for parsing and generation – basically we use the Earley deduction proof procedure as introduced in [Pereira and Warren, 1983] – but to use a data-driven selection function, in the sense that the element to process next is determined on the basis of the current portion of the input. In the case of parsing this is the string, and for generation it is the semantic expression.

This enables us, for example, to obtain a left to right control regime in the case of parsing and a semantic head driven regime in the case of generation when processing the same grammar by means of the same underlying algorithm.

A second novel idea of the new approach is to use an *uniform indexing mechanism* for the retrieval of already completed subgoals (i.e., lemmas). It is uniform in the sense that the same basic mechanism is used for parsing and generation, but parameterized with respect to the information used for indexing lemmas. More precisely, in the case of parsing, lemmas are indexed using string information and in the case of generation semantic information is used to access lemmas. The kind of index causes completed information to be placed in the different state sets. Using this mechanism we can benefit from a table-driven view of generation, similar to that of parsing. For example, using a semantics-oriented indexing mechanism during generation massive redundancies are avoided, because once a phrase is generated, we are able to use it in a variety of places.

Based on the uniform indexing mechanism we present a novel method of grammatical processing which we call the *item sharing* method. The basic idea here is that partial results computed during one direction (e.g., parsing) are automatically made available for the other direction (e.g., generation), too. Since now items are shared by both directions we call them *shared items.* We show how the uniform tabular algorithm is easily extended to make use of shared items. The usefulness of the item sharing approach will emerge when the method of interleaving parsing and generation is introduced. We demonstrate that both the uniform tabular algorithm and the item sharing method make the exploration of such an approach a worthwhile venture.

Furthermore, the uniform tabular algorithm is embedded into an agenda control mechanism, such that new lemmas are first inserted into an agenda. The agenda is structured as a priority queue to store lemmas that have not yet been added to the table. Since lemmas are added to the table according to their priority, we can easily model depth-first, breadth-first and even preference-based strategies.

These aspects together allow us to consider parsing and generation as the same uniform process which is capable of efficiently controlling the space of possible constructions in a *task specific data-oriented manner.* On one hand this enables us to obtain a stronger goal-directed generation behaviour as the one proposed by [Shieber *et al.*, 1989], by taking advantage of the Earley deduction method. On the other hand we are able to characterize parsing and generation in a fairly balanced way without the loss of efficient properties. Hence, we avoid the complications or restrictions that [Shieber, 1988] and [Gerdemann, 1991] are confronted with, because of their "parsing oriented" view of generation. Moreover, the new uniform tabular algorithm together with the item sharing approach makes interleaving of parsing and generation not only plausible but also practical. Furthermore, because we can easily adapt our uniform algorithm for more sophisticated control techniques (e.g., the use of preferences) it is more amenable to future refinements than most of the algorithms described so far.

Since the only relevant parameter our algorithm has with respect to parsing and generation is the difference in input structures – a string for parsing and a semantic expression for generation – the basic differences between parsing and generation are simply the different input structures. This seems to be trivial, however our approach is the first uniform algorithm that is able to adapt itself dynamically to the data, achieving *a maximal degree of uniformity for parsing and generation under a task-oriented view.*

Overview

This chapter is organized as follows. In the next section 4.1 we give a short overview of Pereira and Warren's original Earley deduction scheme and introduce some basic notations. In sections 4.2 through section 4.6 we present a first version of the uniform tabular algorithm and give a parsing and generation example in section 4.7.

Although this version already realizes novel ideas (e.g., data-driven selection function, a new top-down semantic-driven generation strategy) it is not as efficient as it could be. Therefore we present in sections 4.8 and 4.9 a uniform but data-driven indexing mechanism that is used for efficient retrieval of re-solved lemmas. In section 4.10 we describe the necessary modifications of the uniform algorithm to be able to make use of the uniform indexing mechanism and give a parsing and generation example in section 4.11.

In the section 4.12 we show how to deal with verb second problems. After the description of important properties of the uniform algorithm in section 4.13 and of the implementation of the algorithm in section 4.14, we present the item sharing approach in section 4.15. We first present informally the basic ideas and then show how the method is integrated within the uniform tabular algorithm, and finally describe important implementational aspects.

Before we go on in actually doing the things just promised we would like to stress, that all methods developed in this thesis have been implemented. In order to abstract away from a concrete implementation using a specific programming language we present the methods in an abstract way (making use of a pseudo programming language). However, this abstract description is made in just a way that the underlying implementation will be transparent. Thus the way we are describing the methods and the way they are implemented converges.

4.1 Overview of Earley Deduction

In this section we informally describe the Earley deduction method introduced in [Pereira and Warren, 1983]. Earley deduction is a proof procedure for definite clauses and is named after Earley's context-free parsing algorithm. As in Earley's original algorithm, in Earley deduction the processing of definite clauses is split into two basic deduction steps or rules of inference, namely *prediction* and *completion*[1], dealing respectively with top-down predictions of new clauses and bottom-up combination of existing clauses. The results of prediction and completion (the *derived clauses*) are lemmas, i.e., logical consequences of the program.

Earley deduction operates on two sets of definite clauses, called the *program* and the *state*. In our case the program just represents the grammar and lexicon and remains fixed. The state set, on the other hand, will be continually augmented with new lemmas. Whenever a new non-unit lemma is added, one of its negative literals is selected (for example, when using the basic Prolog strategy, the selected element would always be the leftmost literal in the body of a lemma).[2]

The prediction rule operates on non-unit lemmas, which we also call *active lemmas*. This rule selects an active lemma B from the current state and searches for a program clause C whose positive literal (i.e., the head of C) unifies with the selected literal of B. If this is possible the thereby derived clause $\phi[C]$ is added to the state set, where ϕ is the most general unifier of the two literals. The selected element of B is said to *instantiate* the program clause C to $\phi[C]$, or in other words, the selected element has been used to predict an instantiation of C. Clearly, prediction thus described realizes the top-down step of Earley deduction.

The completion rule operates on unit lemmas, which we also call *passive lemmas*. An active lemma C is chosen from the current state set, whose selected element unifies with the passive lemma, which (if possible) yields a new resolvent $\phi[C']$, where C' is C minus its selected element. Since the completor actually reduces the number of the negative literals of an active lemma, repeated application of the completion rule eventually creates passive lemmas. Note that completion against the selected literal is sufficient, because if a completion with some other literal of the body is required, any selection function will sooner or later come to this literal, because the selection function will have to select from fewer and fewer literals (see also [Pereira and Shieber, 1987], page 199). Hence, the completor is also said to *reduce* C to $\phi[C']$ or in other words, the completion step (partially) completes derived clauses, in a bottom-up fashion.

---

[1] Pereira and Warren call these phases *instantiation* and *reduction*, respectively. Note that the scanning operation, known from Earley's algorithm can be seen as a special completion step, namely the completion with an lexical entry, i.e., a unit clause representing a terminal element.

[2] Although, [Pereira and Warren, 1983] abstract away from a specific selection function, they do not suggest how to parameterize the selection in a concrete implementation. Furthermore, they only consider parsing under the paradigm of deduction.

In [Pereira and Shieber, 1987] it is shown that the more specific constraints of context-free parsing allow a simplification that we cannot take advantage of here. In Earley's parsing algorithm, derived clause creation proceeds strictly from left to right. This means that any passive lemma needed to resolve against some active lemma is guaranteed to be constructed after the active lemma is created. Therefore, to perform all the pertinent resolutions, the algorithm need only look for active lemmas at the time when a passive lemma is created. However, a general Earley deduction proof procedure cannot guarantee this, so it is necessary to apply the completor also when active lemmas are created. As we will show, this is specifically the case for generation, because the structure of the input is not a sequence but a tree-like structure. In the implementation described in the next sections, we will separate these two cases into two inference rules, called *passive-completion* and *active-completion*, respectively.

Blocking new lemmas As already suggested, a new lemma is only added to the state set if no subsuming clause exists in the table (i.e., one which is more general than the newly derived clause). If a new lemma cannot be added to the state set because a more general one already exists, then the lemma is said to be *blocked*.

The subsumption check is necessary, because we are dealing with instantiations of clauses, not with the clauses directly (as it is the case for context free versions,where it suffices to check whether the same grammar rule has already been predicted). Hence, to avoid the repeated prediction of the same program clause, we have to check whether a more general prediction has already been made. In this case a newly predicted more specific one can just be ignored. In a similar way, we can take advantage of the subsumption check to avoid redundant re-computation of completed lemmas during the completion step. This means, for instance, that once a passive lemma has been derived it can be used for further completion of different active lemmas, without redundant re-computation (in section 4.8, we will show how we can combine this technique with a clever indexing mechanism that is used for parsing and generation).

Use of a restrictor The prediction rule is used for predicting new instantiations of grammar rules using the selected element of a non-unit lemma. As known from the work of [Shieber, 1985] prediction can lead to arbitrary numbers of consequents through repeated application when used with a grammar with an infinite structured nonterminal domain. For example, if a grammar handles subcategorization with list value features (such as in HPSG or the rule (r_1) of the grammar given in appendix A), then non-termination can arise since the subcategorization rule is able to produce instantiations with successively increasing lengths of subcategorization lists, which do not stand in a subsumption relation. Hence, neither of the lemmas is blocked.

The solution proposed in Shieber's work is to use only a restricted amount of information for predicting the element. Thus before the predictor is evaluated a restrictor function RESCT is applied that computes from the constraints of the selected element a bounded subset of the information of these constraints. The restrictor function serves to specify how much information is to be used in the top-down phase of a uniform algorithm. For example, if we use the identity function, all information would be predicted and if we use simply the constant function yielding the trivial model, no top-down information is used.

In Shieber's original work, a grammar-oriented version of a restriction function is presented, where only those features are predicted which have been chosen by the user as most useful for prediction. Following [Shieber, 1985] restriction is defined on the basis of a given restrictor R, which is a finite set of paths through a feature structure. The *restriction* of a feature structure F relative to R is the most specific feature structure $F' \sqsubseteq F$, such that every path in F' has either an atomic value or is an element of R.

A more general definition of RESCT is given in [Haas, 1989]. In this approach, for a given set of constraints ϕ a restricted form ϕ' is computed by replacing all cyclic constraints with new variables. Applying this definition to the subcategorization rule would break the cycle (caused by the variable Tail) of the subcategorization feature SC.[3]

In [Shieber, 1989] it is shown that any function can be used, with the prerequisite that the range of the function is finite. Then termination of prediction can be guaranteed because RESCT divides an infinite number of categories into a finite number of equivalence classes. Thus, after a finite number of applications of prediction, a previously generated item would be built and the subsumption check would prune further applications [Shieber, 1989]. Using a restriction of a feature structure instead of the original feature structure weakens the predicting power of the top-down prediction step in the sense, that it can over predict lemmas. However, it does not affect the correctness of the algorithm, since these unnecessary prediction will never be completed.

In our system we use a restrictor function similar to that of [Shieber, 1985]. However, instead of specifying which constraints should be used to build a restrictor, we specify which information of a specified set of features should be ignored (i.e., set to the top variable). For the grammar in appendix A this is basically the subcategorization feature SC. Thus, before the predictor is applied, then if this feature is present its value is set to the top variable. Any other information is unchanged. In this way, we use as much top-down information as possible, however by being able to ignore "dangerous" features.

[3] In [Samuelsson, 1994] an alternative approach is taken for deriving restrictors automatically by making use of anti-unification (also known as generalization). Anti-unification is the dual of unification – it constructs the least general term that subsumes two given terms. In the case of restriction it is also used for detecting cyclic constraints.

4.2 Generalizing Pereira and Warren's Earley Deduction Scheme

We now present an interpreter for definite clauses that performs according to Earley deduction. Instead of collecting all grammar rules and lexical entries in one data structure, we will use two separate data structures G, which keeps all grammar rules and Lex, which holds all lexical entries.

PREDICTION is used to predict instantiations of grammar rules. Completion will be performed by three inference rules, namely PASSIVE-COMPLETION, ACTIVE-COMPLETION, and SCANNING. In all three cases, unit clauses will be used to reduce appropriate non-unit clauses, where the scanning rule can be seen as a special active-completion rule in the sense, that is looks for unit clauses of the lexicon which it uses to reduce the non-unit clause in question.

We will use S to denote the state set, to which new lemmas are dynamically added. However, instead of directly storing lemmas into the table, new lemmas are stored in an agenda where the lemmas are sorted according to a given priority. Thus lemmas are added to the table according to their priority. However, since the inference rules can generate new lemmas which are inserted to the agenda before they are added to the table, we can use the agenda mechanism to model depth-first, breadth-first and even best-first strategies. Another advantage of using an agenda mechanism, is that the subsumption test to see whether an item is already in the table, is needed only when the item should be added to the table. In the case, where we only want to yield one result instead of all possibilities, the subsumption test has to be performed only for those items which are actually added to the table. Thus, if we follow a depth-first strategy the number of items to be inserted into the table is less than those which had been inserted to the agenda (if we are only interested in one result).

4.3 A Data-driven Selection Function

The discussion of current approaches for parsing and generation can be summarized as follows: parsing and generation, to be goal-directed, differ basically with respect to the order in which the literals of the body of a clause are selected. For parsing, for example, [Shieber, 1988; Gerdemann, 1991] have used the leftmost selection strategy, where for generation [Shieber *et al.*, 1989; Gerdemann, 1991] use the semantic-head first selection function. The latter should be seen more precisely as a "preference-based" selection function, since in the case a rule has no semantic head, the leftmost element is chosen, or if two elements share the semantics with the mother node, the left one is selected.

However, it is very easy to combine these different strategies used in parsing and generation, such that the selection function expresses a preference for goals with certain features instantiated.[4] Since we want to obtain an input driven algorithm, the essential feature for parsing should be the PHON path (more precisely the path is $\langle$PHON DL$\rangle$, that is the path to the list value of the difference list) and for generation it should be the SEM path. We will call this certain feature the *essential feature* Ea. Then the selection function can be defined such that it selects the leftmost element from the body whose essential feature is instantiated, i.e., whose Ea exists and is a non variable value. If such an element does not exist it chooses the leftmost element. Now, if we abstract away from a concrete essential feature by assuming that Ea is a variable, then we can define this selection function more formally as follows:

$$\textsc{sf}(q \leftarrow p_1, p_2, \ldots, p_i \ldots p_n, Ea) = \begin{cases} i & p_i, \text{ the first element whose Ea is instantiated} \\ 1 & \text{otherwise} \end{cases}$$

Now, in order to use this selection function for parsing or generation we have to specify a path that defines the essential feature (i.e., the phonological or semantic path). Since, the value of this feature will be a string or semantic expression, this means that the selection function prefers those goals which are instantiated with a string or semantic expression. However, now, the grammar itself will be an important source of control, since it defines how information is decomposed (or composed depending on the point of view) in the rules. For example, if the phonological information is expressed as difference lists and partial strings are combined by string concatenation then the selection function SF “realises” a leftmost strategy. Similarly, if all rules define a semantic head relation SF simulates the semantic head first relation. These can both be true at the same time. Moreover, if the grammar rules are attached with some preference values, the selection function can very easily be adapted to take into account such preference information. This would help to achieve a more careful selection.

4.4 A Data Structure for Lemmas

We will use the following notation for an active lemma and its selected element:

$$\langle h \leftarrow b_0 \ldots b_n \ ; \ i \rangle$$

[4] Most recently, [Johnson, 1993] also presents a deduction mechanim which makes use of a dynamic selection function. The basic use of this selection function, however, is to support a coroutine-oriented selection between the body elements of a clause. Since, he only focused on natural language parsing, we do not consider this method in more detail.

One should also keep in mind, that [Pereira and Warren, 1983] already abstracted away from a specific selection function. For example, they already outlined the idea of a head-oriented selection function for parsing. This is important to say, because often their approach is viewed to be restricted only for a left-to-right scheduling.

where $h \leftarrow b_0 \dots b_n$ is an active lemma and i $(0 \leq i \leq n)$ is the index of the selected element in the body of the lemma. We will call such structure an *active item.* The structure of unit lemmas will be represented as:

$$\langle h \ ; \ \epsilon \rangle$$

where ϵ indicates that since the body of the lemma is empty the selected element is also. We will call such structures *passive items.*

When new lemmas are generated, the actual selection function determines the next element to process.[5] This is either an element of the body of a lemma in the case the new lemma is non-unit, or ϵ for unit lemmas.

Specification of Goals

As already noted in chapter 3 a query p as input for parsing or generation is a possibly empty conjunction of an $\mathcal{R}(\mathcal{L})$-atom and a $\mathcal{L}$-constraint, written as:

$$\leftarrow p, \phi$$

Usually, the query corresponds to the root node of the grammar, where the constraints include at least the string to parse or the semantics to generate. For example, using the lexical grammar of appendix A for parsing the sentence "weil peter lügen erzählt" the goal statement would be

$$\leftarrow sign \left(\begin{bmatrix} \texttt{phon} & \langle weil, peter, lügen, erzählt \rangle \text{-} \langle \ \rangle \\ \texttt{cat} & comp \end{bmatrix} \right)$$

while the goal statement for generating this string would be

$$\leftarrow sign \left(\begin{bmatrix} \texttt{cat} & comp \\ \texttt{lf} & \begin{bmatrix} \texttt{type:} & unary \\ \texttt{pred:} & weil \\ \texttt{arg:} & \begin{bmatrix} \texttt{type:} & binary \\ \texttt{pred:} & erzählen \\ \texttt{arg1:} & \begin{bmatrix} \texttt{type:} & nullary \\ \texttt{pred:} & peter \end{bmatrix} \\ \texttt{arg2:} & \begin{bmatrix} \texttt{type:} & nullary \\ \texttt{pred:} & lügen \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right)$$

However, in order to bind the resulting constraint so that we can easily return it as a value from the underlying proof procedure, we specify the goal statement as the negative literal of an $\mathcal{R}(\mathcal{L})$ -atom that does not belong to the grammar or lexicon. The constraints of the new atom will be the same as the goal statement. Thus for the example above the goal statement will be specified as[6]

[5] To be more precise, the selection function returns the index of the selected element. As long as no misunderstandings are possible, we will use selected element and index of selected element in the same sense.

[6] Pereira and Warren also specify goal statements in such a way, eventually because of the same reason.

$$ans\,(Fset) \leftarrow sign\left({}_{Fset}\begin{bmatrix}\texttt{phon} & \langle weil, peter, l\ddot{u}gen, erz\ddot{a}hlt\rangle\text{-}\langle\ \rangle \\ \texttt{cat} & comp\end{bmatrix} \right)$$

If this clause can be reduced, the constraints on the head will also express the resulting feature structure which then can easily be returned as output from the proof procedure.[7]

The general structure of such item is:

$$\langle ans\,(L) \leftarrow p\,(L)\,; \textsc{sf}(ans\,(L) \leftarrow p\,(L)\,, Ea)\rangle$$

where p is a predicate of the grammar, ans a predicate not in the grammar, $\textsc{sf}(ans\,(L) \leftarrow p\,(L)\,, Ea)$ the index of the selected element. We will denote such item as the *start item*.

4.5 The Inference Algorithm

Instead of giving the definition in terms of a concrete programming language, we will use simple pseudo programming language encoding.[8]

The body of a program will be specified by using well-known logical operators. The input and output parameters of a procedure are specified using the keywords **in** and **out**. For example the notation

FIRST(**in:** LIST; **out:** ELEMENT)

means that the input parameter of the procedure FIRST will be bound to the formal parameter LIST and its output value will be bound to ELEMENT. Procedure names are represented in small caps and primitive operations of the pseudo language are highlighted using the boldface font (e.g., **if**, **then**, **else**, **while**). Capitalized strings are used to denote variables.

[7] Otherwise, we have to store separately the feature structures of a resolved query, because a resolved (and hence reduced) query would just be the clause of the form

$$\epsilon \leftarrow \epsilon, \phi$$

Note that using the ANS atom is only a technical matter and it should be interpreted as a "special empty head of a clause." However, since we want to stress also important implementational aspects of our algorithm, we have decided to make our implementation as transparent as possible.

[8] It would also be possible to define the inference rules more abstractly in terms of logical inference rules as for example done by [Shieber, 1988]. However, we are interested in algorithmic and data structure aspects, since this will be important for the next chapters to come. Hence, we prefer a more programming language oriented specification of the inference rules.

The result of each inference rule (i.e., the new items) will be added to an agenda using the function ADD-TASK-TO-AGENDA. We will assume a global variable Agenda which is bound to the current agenda. Which priority is determined for a new item is computed by the procedure PRIO. The agenda control is then responsible for adding new items to the state set S according to a given priority. However, a new item is only added to S, if it is not blocked. We will abstract away from a concrete definition of this procedure as well as from the embedding of the inference rules until the agenda-based control mechanism is introduced below.

All inference rules will receive as input an item. We will assume that the global variable Ea is bound to the actual essential feature. The result of each inference rule will be either **true** of **false** depending on whether new items could be derived or not. We will indicate this by a boolean variable Candidates?. Its initial value is **false** and will only be set to **true** if a new item is added to the Agenda.

Prediction We start by defining the prediction rule. Figure 4.1 specifies the procedure.

```
(1) proc          PREDICTION(in: AL; out: Candidates?):
(2) Candidates?   := false;
(3) ∀ Rule ∈ G;   Φ = UNIFY(RESCT(SEL(AL)), HEAD(Rule)) and Φ ≠ ⊥
(4) do            NewItem := ⟨Φ[Rule]; SF(Φ[Rule],Ea)⟩;
(5)               ADD-TASK-TO-AGENDA(NewItem,PRIO(NewItem),Agenda);
(6)               Candidates? := true;
(7) od.
```

Figure 4.1: The procedure for prediction.

Prediction will be performed on active items AL and will return **true** or **false** depending on whether a new item has been added to the agenda (see line (1)). It will predict instantiations Φ[Rule] of all rules Rule in the grammar G whose head HEAD(Rule) unifies with the restriction of the selected element of the active item, denoted as RESCT(SEL(AL)) (line (3)) and inserts new items (line (4)) into the agenda Agenda according to a given priority determined by the function PRIO individually for each new item (line (5)).

Completion As already noted in 4.1, completion will be split into three separate rules, namely *passive-completion*, *active-completion*, and *scanning.*

Passive-completion (see figure 4.2) will be applied on passive items PL. For all active items AL in S which selected element SEL(AL) unifies with the head of the passive item, the new lemma Red is Φ[AL-SEL(AL)], i.e., the unified item AL minus its selected element (line (4)). The new item NewItem (line (5)) is added to agenda Agenda (line (6)). Since passive-completion reduces the body of an active lemma

```
proc            PASSIVE-COMPLETION(in: PL; out: Candidates?):
Candidates?     := false;
∀ AL ∈ S;       Φ = UNIFY(SEL(AL),HEAD(PL)) and Φ ≠ ⊥
do              Red := Φ[AL-SEL(AL)];
                NewItem:= ⟨Red; SF(Red,Ea)⟩;
                ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
                Candidates? := true;
od.
```

Figure 4.2: The procedure for passive-completion.

it will transform it to a passive lemma by repeated application. Furthermore, it performs the completion step bottom-up.

```
proc            ACTIVE-COMPLETION(in: AL; out: Candidates?):
Candidates?     := false;
∀ PL ∈ S;       Φ = UNIFY(SEL(AL),HEAD(PL)) and Φ ≠ ⊥
do              Red := Φ[AL-SEL(AL)];
                NewItem := ⟨Red; SF(Red,Ea)⟩;
                ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
                Candidates? := true;
od.
```

Figure 4.3: The procedure for active-completion.

Active-completion (see figure 4.3) is very similar to passive-completion. The basic difference is that active-completion will receive an active lemma AL and will search S for passive-lemmas PL that can reduce AL. Thus, active-completion performs the completion step in a top-down manner. Again, active-completion can transform an active lemma to a passive lemma by repeated applications.

```
proc            SCANNING(in: AL; out: Candidates?):
Candidates?     := false;
∀ LE ∈ Lex;     Φ = UNIFY(SEL(AL),HEAD(LE)) and Φ ≠ ⊥
do              Red := Φ[AL-SEL(AL)];
                NewItem := ⟨Red; SF(Red,Ea)⟩;
                ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
                Candidates? := true;
od.
```

Figure 4.4: The procedure for scanning.

Scanning has mainly been described from a parsing perspective in the sense, that it is limited to only scanning words that match the initial portion of the input. However, when the scanner should also be usable for generation it must be generalized such that it should be allowed to consider any lexical entry that will unify with the selected element of an active item. In a constraint-based approach the input is specified as part of the constraints. This allows us to describe scanning as a specific active-completion rule, namely the completion of an active item with possible matching unit clauses from the lexicon.

The scanning rule (see figure 4.4) operates on a given active item AL and returns the boolean value of Candidates?. The scanner searches through the lexicon to look for lexical entries LE whose HEAD(LE) unifies with the selected element of the active item, denoted as SEL(AL) (line (3)). For all successful instantiations, the new lemma Red is Φ[AL-SEL(AL)], i.e., the unified item AL minus its selected element is constructed (line (4) and (5)), and is then added as a new task to the agenda. Thus, scanning also reduces an active item. Since, the selected element has been determined on the basis of the value of the essential feature, this also means that the scanner will consume portions of the input on the basis of available and consistent lexical information.

We have not explicitly required that scanning should only be performed on terminal elements, i.e., active items, whose selected element belongs to a terminal category. The reason is, that in general constraint-based grammars are under-specified in this respect. For instance, in our example grammar some of the lexical and phrasal signs belong to the same category (e.g., vp, np) and they are only distinguished by the boolean feature LEX. However, most of the rules are under-specified with respect to this information. In that case we have to apply scanning and prediction on the same active item. Of course, if a grammar explicitly distinguishes between nonterminal and terminal elements (as it is the case for instance in LFG), we can easily restrict the application of the scanning rule to terminal elements and the prediction rule to nonterminal elements.

4.6 An Agenda-based Control Regime

The inference rules will be embedded in an agenda-based control regime along the line of [Shieber, 1988]. An agenda consists of a list of *tasks* and a policy for managing it. A task is simply an item. Whenever an inference rule creates a new item it is added as a new task to the agenda and sorted according to the given priority function PRIO. If the agenda chooses a task as the next item, and this item is not blocked, it is added to the state set. Using the agenda mechanism in this way has the effect that the subsumption operation (performed inside the blocking test) is delayed until an item is explicitly chosen by the agenda control. If our agenda control follows, for example, a depth-first strategy, then only "useful" items are considered. In chapter 5 we show the importance of this behaviour for the case of performing self-monitoring and revision during natural language generation.

We will assume the following operations to be defined for agendas:

MAKE-AGENDA() creates a new agenda.

EMPTY-AGENDA-P(Agenda) is true if the specified agenda is empty.

GET-HIGHEST-PRIO-TASK(Agenda) removes and returns the task with the highest priority.

ADD-TASK-TO-AGENDA(Tasks,Prio,Agenda) adds new Tasks with priority Prio to the specified Agenda.

PRIO(Task) determines the priority for the task Task.

The particular agenda that we use is denoted by the global variable Agenda. The control regime denoted as PROCESS will manage the agenda. During the course of PROCESS it dequeues tasks from the agenda until the agenda is empty. However, note that the inference rules force the agenda to add new tasks.

Each dequeued task is added to the table S. This operation is performed by the function ADD-ITEM. However, ADD-ITEM only adds a new item to the table if it is not blocked. In that case the task (that actually has been become a real item) is passed to the procedure APPLY-TASK which applies the inference rules that eventually cause the creation of new tasks.

The procedure PROCESS (see figure 4.5) receives as input the goal Goal to prove and returns as a result either rejection or a list of answers (line (1)). It first creates a start item StartItem using the function MAKE-START-ITEM and adds it to the Agenda. Then, while the Agenda is not empty (i.e., there are still tasks waiting to apply), GET-HIGHEST-PRIO-TASK determines the next task to process (and removes it from the agenda). If the new CurrentTask is not blocked by some item already in the table S it is added to S (this test and operation is performed by the function ADD-ITEM, see figure 4.6). In that case the new Task is applied on the inference rules (which is performed by the function APPLY-TASK). Note that the inference rules called by this procedure will eventually create new tasks. If the state set contains a new answer this is added to the Result list.[9]

[9] In our implementation, we distinguish between new and old answers by means of an additional slot attached to an item named IGNORE, which is set to true, if an answer is pushed to the Result list. Then, during further processing this item is ignored. In a sense, the item has become garbage and could also be removed from the state set, by a kind of garbage collector for items.

```
proc        PROCESS(in: Goal; out: Result):
StartItem   := MAKE-START-ITEM(Goal);
ADD-TASK-TO-AGENDA (StartItem,PRIO(StartItem),Agenda);
while       NOT(EMPTY-AGENDA-P(Agenda))
do          CurrentTask := GET-HIGHEST-PRIO-TASK(Agenda);
            if ADD-ITEM(CurrentTask) then
            do APPLY-TASK(CurrentTask)
            if ∃ I ∈ S; I is of form ⟨ans; ε⟩ then
            Result := push(I,Result); fi
            od
od
Result :=   if empty(Result)
            then rejection
            else Result fi.
```

Figure 4.5: An agenda-based control mechanism.

```
proc   add-item(in: Task; out: Candidate?):
if     Task is not blocked in S then
do     ADD Task to S;
       Candidates? := true;
od     else Candidates? := false;
fi.
```

Figure 4.6: The procedure that adds new items to the table if they are not blocked.

The procedure APPLY-TASK (see figure 4.7) applies the inference rules to the task Task (line (1)). If the current task is a passive item (line (2)) PASSIVE-COMPLETION is called (line (3)). Otherwise (line (4)) it is checked whether ACTIVE-COMPLETION returns **true**, which means that new items have been added to the agenda. If this is not the case PREDICTION and SCANNING are called. The reason why we only consider prediction and scanning if active-completion returns **false** (i.e., creates no new task) is that if active-completion is successful this means that for the selected element of the current active item there already exists a derived phrase (made for the same substring or partial semantics) and hence, prediction and scanning would be redundant.

Scheduling the tasks of an agenda The agenda method is based on a priority queue. The elements in the queue are ordered relative to their assigned priorities, where the task with the highest priority is at the front of the queue and the one with the smallest is at the back. Using this mechanism it is very easy to realize a *depth-first* or *breadth-first* strategy. Suppose that we maintain a counter LemmaCounter

```
proc   apply-task(in: Task):
if     PASSIVE-ITEM(task)
       then        PASSIVE-COMPLETION(Task) else
                   ACTIVE-COMPLETION(Task) else
       do          PREDICTION(Task);
                   SCANNING(Task);
       od fi
fi.
```

Figure 4.7: The procedure APPLY-TASK.

that enumerates the stored lemmas starting from one. Directly using the value of this counter realizes a depth-first strategy, since each new task is added to the front of the queue. Using its negation instead would realize a breadth-first strategy, because processing of new items is delayed until older tasks are processed.

The advantage of a depth-first strategy is that in the case only the first result should be computed the number of items added to the table S is less than the number of generated items added to the agenda. Since, only in the case an item is added to the table the blocking rule is applied, the subsumption operation need only be performed on a subset of the items stored in the agenda. This means, that although our inference rules exhibit a breadth-first component, using an agenda mechanism restricts the computional overhead of subsumption on that set of items that are used in specific situations, e.g., depth-first and best-first strategies.

In addition to a depth-first and breadth-first task-selection function, we have also defined a definition for PRIO where the priority is determined randomly using a built in function RANDOM, because all three priority functions together characterize a representative degree of possible agenda strategies.

However, it would also be possible, to use more complex priority functions as for instance the one proposed in [Shieber, 1988]). This would open up the realization of some psychologically interesting strategies (like garden-path phenomena, minimal attachment phenomena, or more general preference based strategies, see [Kay, 1986; Shieber, 1988; Görz, 1988; Shieber, 1989; Erbach, 1991]), however using the same deductive approach. It should be clear, that our specification is open to such more sophisticated use of priority functions.

4.7 Performing Parsing and Generation

Parsing is defined as a specific call to the main procedure PROCESS, by specifying a concrete value for the global variable Ea. We will use the path $\langle$ PHON DL $\rangle$. The parser is then just a call to PROCESS.

(1) **proc** PARSE(**in:** Goal; **out:** Result):
(2) Ea := ⟨ PHON DL ⟩;
(3) Result := PROCESS(Goal)

When called with a goal Goal it simply returns the result of the procedure PROCESS. For example (using the grammar in A, for parsing the string "weil peter heute lügen erzählt" the input for the parser is

$$\longleftarrow sign\left({}_{G}\begin{bmatrix} \texttt{cat} & comp \\ \texttt{phon} & \langle weil, peter, heute, lügen, erzählt\rangle\text{-}\langle\ \rangle \end{bmatrix} \right)$$

and the created start item is

$$\langle\ ans(G) \longleftarrow sign\left({}_{G}\begin{bmatrix} \texttt{cat} & comp \\ \texttt{phon} & \langle weil, peter, heute, lügen, erzählt\rangle\text{-}\langle\ \rangle \end{bmatrix} \right)\ ;\ 0\ \rangle$$

Figure 4.8 shows a trace of the parsing process. The trace does not give a full example run, because we only considered those steps in the proof which contribute to the resulting derivation shown in figure 4.9. P stands for prediction, S for scanning and C for completion. The arrows basically represent the flow of information. The notation (i), where i is an integer denotes the computation step, e.g., (1) is the first step and (13) the last one. For simplicity, we have only expressed the phrasal backbone and the input string (also simplified). Although the subcategorization rule has been predicted two times, this prediction has been done on different string positions. However, this rule will only be predicted one time at the same string position. Thus left-recursion is no problem. Note that we only show the successful steps.

The generation instance GENERATE differs only with respect to the specified value for Ea, and that the goal specifies the semantic form for which the program should generate corresponding strings. Thus, the generation instance is as follows:

(1) **proc** GENERATE(**in:** Goal,; **out:** Result):
(2) Ea:= SEM;
(3) Result := PROCESS(Goal).

As an example consider generation from the start item

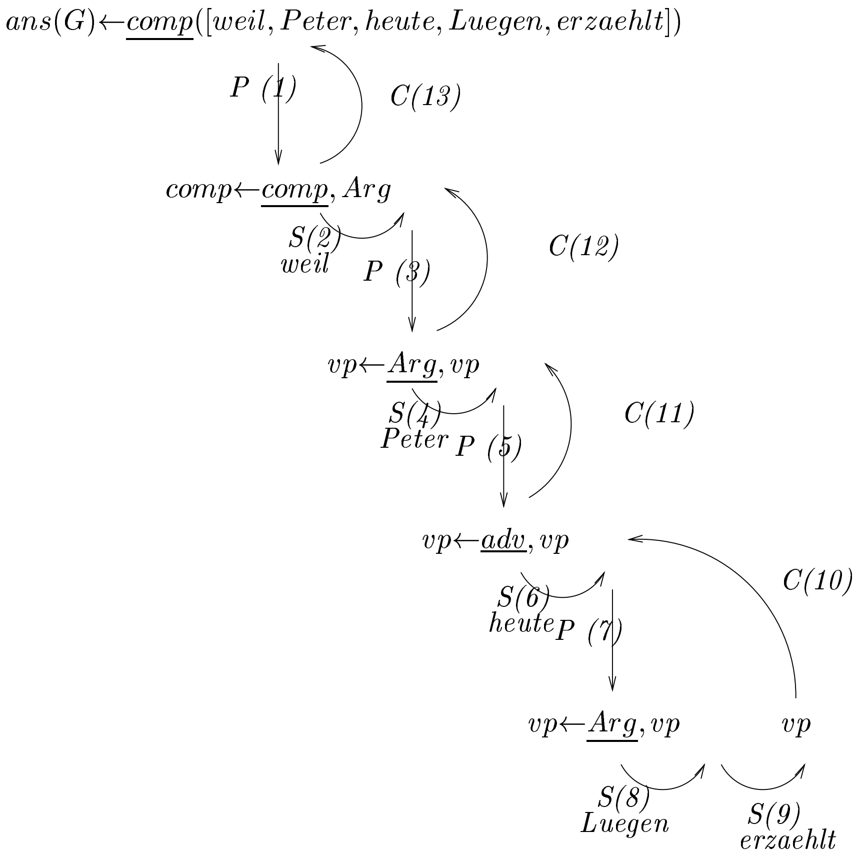


Figure 4.8: A trace of parsing the string "weil peter heute lügen erzählt." For explanations of the symbols used see text.

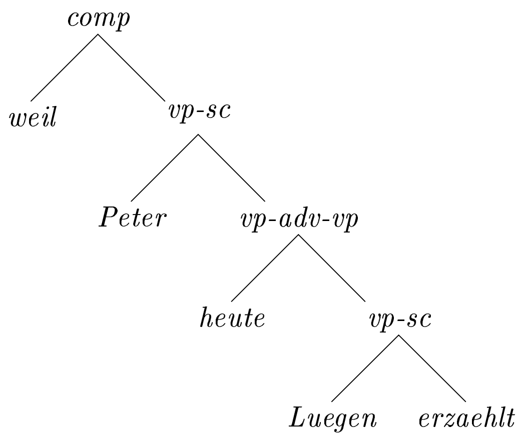


Figure 4.9: The derivation tree of the example sentence. The labels of the node refer to the names of the rules of the grammar in Appendix A.

$$\langle\ ans(G) \longleftarrow sign\left({}_{G}\begin{bmatrix} \texttt{cat:} & comp \\ \texttt{sc:} & \langle\ \rangle \\ \texttt{sem:} & \begin{bmatrix} \texttt{type:} & unary \\ \texttt{pred:} & weil \\ \texttt{arg:} & \begin{bmatrix} \texttt{type:} & unary \\ \texttt{mod:} & heute \\ \texttt{arg:} & \begin{bmatrix} \texttt{type:} & binary \\ \texttt{pred:} & erz\ddot{a}hlen \\ \texttt{arg1:} & \begin{bmatrix} \texttt{type:} & nullary \\ \texttt{pred:} & peter \end{bmatrix} \\ \texttt{arg2:} & \begin{bmatrix} \texttt{type:} & nullary \\ \texttt{pred:} & l\ddot{u}gen \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right)\ ;\ 0\ \rangle$$

Figure 4.10 shows a trace of the generation instance. Instead of directly using the feature structure representation of the semantic input we simply use the tree-like list-notation "weil(heute(erzählen(peter,lügen)))". We furthermore made use of the same abbreviations as in the parsing trace of figure 4.8. Note that in step (6) there is already a scanning step possible. The lexical entry can complete the subcategorization rule and the new item (a reduced copy) is used for scanning the next possible entry (step (7)). After completion of the rule, this reduced passive item can be used to complete the predicted subcategorization rule. Thus, although the subcategorization rule has only be predicted one time, it has been used by the completor two times. Thus non-termination because of prediction of infinite long subcategorization lists is avoided.

The derivation of the anlaysis of this semantic input is shown is figure 4.11

tree a. The other two derivations are also possible for this input, but we have not specified the traces here. However, when these two other alternatives are generated precomputed substrings will be reused.

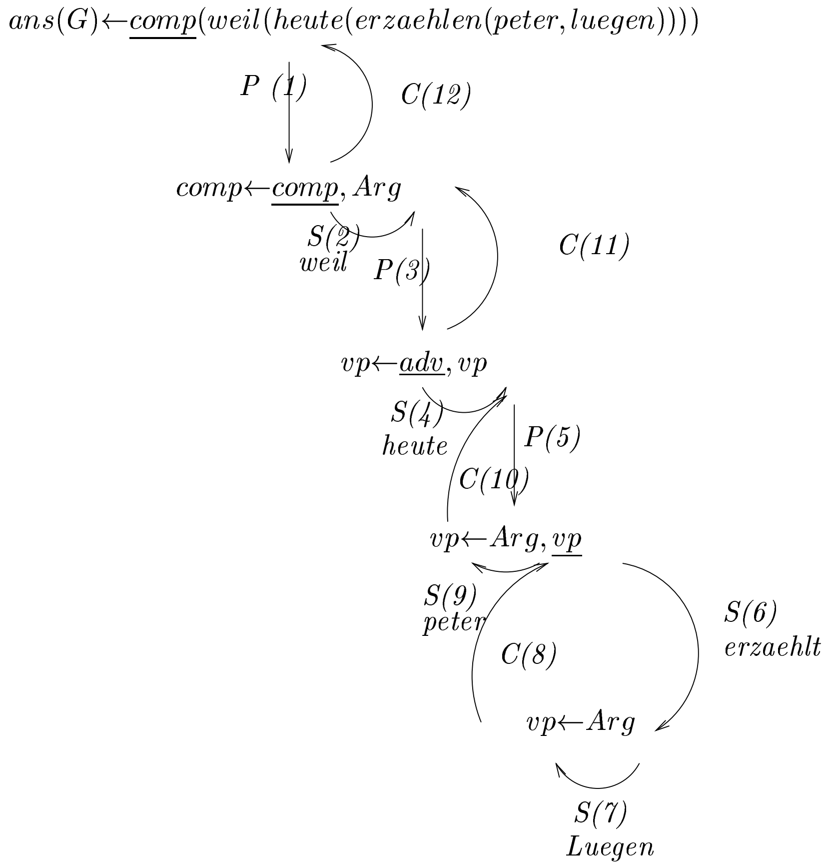


Figure 4.10: A trace of generating from "weil(heute(erzaehlen(peter,luegen)))". For explanations of the symbols used see text.

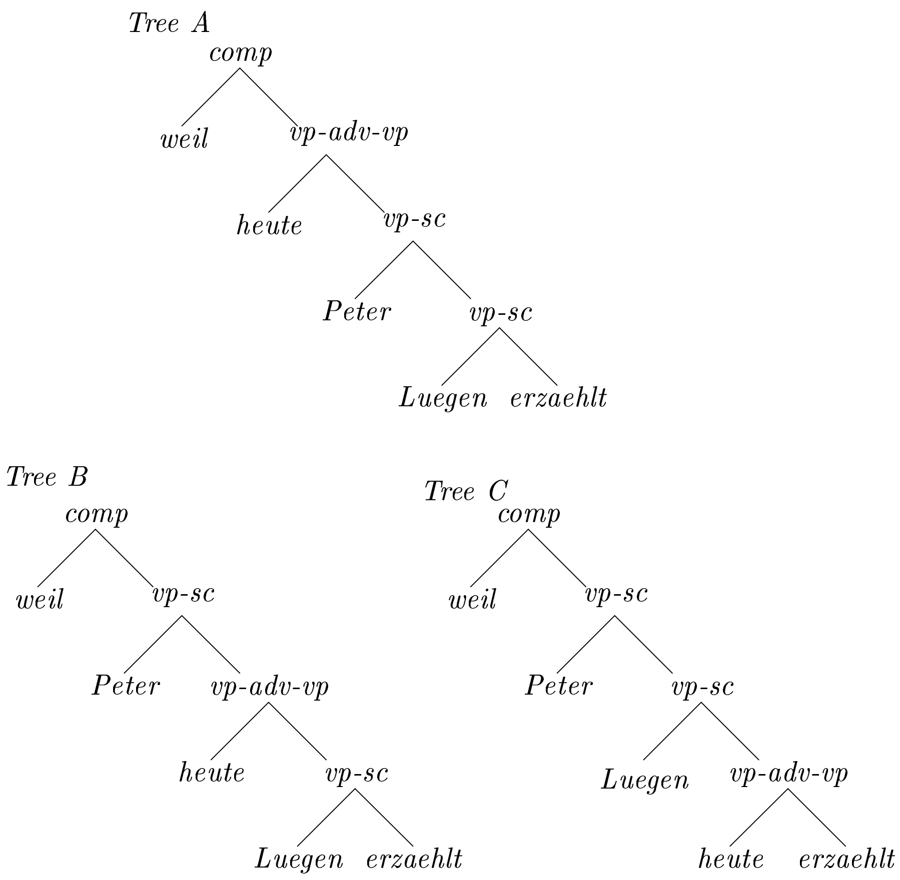


Figure 4.11: All possible derivation trees admitted by the grammar for the generation example.

4.8 Indexing Derived Clauses

Although the tabular version described above already avoids redundant recomputation because phrases just analysed (i.e., parsed or generated) are stored in a table, the search for derived lemmas to resolve is not as efficient as it could be. The problem is that all lemmas are kept in one state set only. For the completor, this causes an enormous overhead, since the whole state set has to be searched through to look for active lemmas which can be reduced. In general, this causes a large number of nonproductive (nonunifying) attempts at applying the completion rule. Furthermore, for each added item the blocking rule has to be performed on the whole set of items already in S.[10]

In order to overcome these disadvantages, the items in S should be ordered according to the structure of the actual problemsize (either string or semantics) such that the inference rules (and the blocking rule) need only be applied to an identifiable subset of the states in each state of the process. The different states of the process can then be defined relative to subsets of items. In this sense, the progressing process decomposes the state set into a set of *internal item sets*, which stand in a relationship according to the structure of the problem size.

For parsing, particular data structures have been developed for achieving such an efficient behaviour, most notably the *chart* developed by [Kay, 1986] and the *item set* notation developed by [Earley, 1970]. In both approaches the endpoints of a derived string are explicitly used for indexing stored phrases. Unfortunately, we cannot use these well-known approaches for generation directly, because the string is the output of a generator, not the input, of course. For generation, once a phrase has been constructed, we want be able to use it at various places.

We now present an indexing mechanism that can be used in the same manner for both parsing and generation. However, since we use the value of the essential feature for determining the "content" of internal item sets, the item sets are ordered according to the actual structure of the input. Note that only the selection function and this indexing mechanism have to be parameterized. However, since the only parameter is a certain feature and its value we had achieved *a maximal degree of uniformity for parsing and generation under a task-orentied view.*

4.9 A Uniform Indexing Mechanism

The basic idea is to split the generated items into equivalence classes and to connect these classes, so that each item can directly be restricted to those items that belong

[10] Clearly, this is the worst case. In our implementation, items are stored in hash tables indexed by the lemma's head's predicate name. However, this is only adavantageous if the grammar uses category specific rules (as in DCG). If only schematic rules are used (as it is the case for the grammar in Appendix A), then hashing in the described way would not help much, since all lemmas have the same predicate name.

to a particular equivalence class. We will call each equivalence class an *item set*. The whole state set then consists of a set of item sets, which we will call a *chart*. We will use the current value of the essential feature Ea of some goal G (abbreviated as $\text{VAL}(G/Ea)$) as index for an item set.

We then require that for each item L in an item set I with index Idx, that $\text{VAL}(L/Ea)$ must be the same as Idx. However, we have to distinguish between passive and active items. We require that for passive items, the essential feature's value is determined from the feature structure of the head element. Clearly this is the only possibility, since passive items have an empty body. For active items, however, we require that the essential feature's value is determined by the selected element.

More formally, we can define an item set I as a tuple $\langle AL, PL, Idx\rangle$, where PL is a finite set of passive items and AL a finite set of active items such that:

$$\forall\, pl_i, pl_j \in PL : \text{VAL}(pl_i/Ea) \;=\; \text{VAL}(pl_j/Ea) \;=\; Idx \text{ and}$$

$$\forall\, al_i, al_j \in AL : \text{VAL}(\text{SEL}(al_i)/Ea) \;=\; \text{VAL}(\text{SEL}(al_j)/Ea) \;=\; Idx$$

Thus all items in one item set share one common property, namely that they are compatible with respect to the value of the essential feature of one of their literals, which is the head in the case of an unit lemma, and the selected element in the case of an non-unit lemma.

In this sense, an item set can be viewed as a kind of *meeting place* of active and passive items, such that an active item looks for some passive item to resolve with, and vice versa, that a passive item looks for an active item which it can resolve. However, both are identical with respect to the value of their essential feature. If the result of the reduction operation is a new item, this item will eventually be placed in another item set.

If we start with a start item representing the goal to prove, the start item will be inserted into an item set with an index determined from the value of the essential feature of the query. We will call the item set built from the start item the *initial item set*. We end successfully, if the initial item set entails the reduced goal.

The inference rules will be responsible for creating and maintaining item sets. In order to reuse items that have been predicted in one item set for items in other item sets, we have to link the serval item sets. We will basically use the same mechanism as Earley's algorithm or from Chart parsing, i.e., for each item we keep a backward pointer. The meaning of the backward pointer is the index of that item set from which the initial prediction of some goal was made. If an active item can be reduced, the reduced item inherits the backward pointer from the active item. Thus, an ancestor of an initially predicted item will have the same backward pointer. However, since the indices of item sets are determined on the basis of the value of the essential feature, and the essential feature is the one that carries either a string or semantic expression, the structure of the linked item sets represents the relationship between parts of the input in the way the grammar has decomposed the input.

Before we can clarify this behaviour, we first have to adopt the item structure to the requested information. We add two new slots to an item. A new slot FROM which holds the backward pointer to the item set the item has been predicted from. The slot IN holds the index of the item set of which the item is a member. The general form of an item is as follows:

$$\langle h \leftarrow b_0 \dots b_n \ ; \ i \ ; in; from \rangle$$

Thus each lemma knows from which item set it has been predicted and of which item set it is a member. The values of these slots for a particular item will be accessed by functions having the same name as the slot names. Thus IN(Item) denotes the item set Item is a member of and FROM(Item) denotes the backward pointer.

For active items, the value of the IN slot is just the value of the essential feature of the selected element. If we denote this as ea(i), the general structure of active items is

$$\langle al; \ i \ ; ea(i); ea(k) \rangle$$

where the backward pointer ea(k) means that this active item has been predicted from some active item whose selected element has value ea(k). The general structure of passive items is

$$\langle pl; \epsilon; ea(k); ea(k) \rangle$$

Since the value of the essential feature of the head of the active clause must not change, when reducing its body, ea(k) is just the value ea(head(al)) or ea(head(pl)) in the case of passive items.[11]

But then, passive items in some item set correspond to some grammatically well-formed structure that has been found for ea(k) which corresponds to a part of the whole input. However, we also know that there must be some active item with which the passive item can resolve, since passive items are the result of reduction of some active items.

By definition, these active items and the passive items belong to the same item set. Since active items must originate from top-down prediction of the start item, this means that passive items correspond to valid partial grammatical structure of

[11] However, if we actually inherit the backward pointer from the previous predicted active item, then we may allow, that the string or semantic expression of the predicted active item may be changed. Of course, only if this is done in a consistent manner, we will be able to use a reduced item, i.e., a passive item also for reducing the active item from which the passive item has been predicted.

the input in question. However, this is only true, if the input may not be changed (by deleting parts or by adding parts). But this is exactly what is required by the coherence and completeness condition of the Ea-proof problem.

We will next describe the inference algorithm on basis of this notation, before it is specified in terms of our pseudo programming language.

The indices of the *start item* will be determined from the value of the essential feature of a query. This value will serve as the IN and FROM pointer of the start item. Let q be the query and Ea, the value of its essential feature. Then the start item is as follows (because q is the only element of the body its index is 0):

$$\langle ans \leftarrow q \; ; \; 0 \; ; ea(0); ea(0) \rangle$$

Thus, if we start with a string or semantic expression, and if we can reduce the goal, the initial item set contains an answer of the goal that characterizes the input in question as a valid sign of the language.

The adoption of the four inference rules can be done very similarly. All of them generate either active or passive items depending on the form of the new lemma of the generated item. If the generated item is active, this active item will be added to the item set, whose index is determined on the basis of the selected element of the new item. If such an item set does not exist, it will be created. If the generated item is passive, we add it to the item set which is indicated by the backward pointer of the active item it was generated from. Note that such item set must exist, because otherwise the passive item would not exist.

In all of the inference rules at least active items are involved. They only differ with respect to the second structure involved. For the predictor and scanner this will be the grammar and lexicon. Passive items are involved for passive completion and active completion. Using the notation above, we can describe the inference rules as follows:

Prediction will predict new instantiations of grammar rules on the basis of the selected element of an active item of form

$$\langle h \leftarrow b_0 \ldots b_n \; ; \; i \; ; ea(i); ea(k) \rangle$$

If R is a non-unit unified grammar rule of form $b_i \leftarrow p_1, \ldots, p_m$ with selected element p_j, then generate item

$$\langle R; \; j \; ; ea(j); ea(i) \rangle$$

otherwise (i.e., R is a unit rule) generate item

$$\langle R; \; \epsilon \; ; ea(i); ea(i) \rangle$$

This means that an instantiated empty production will always be placed into the same item set from which prediction took place. Note that the predictor basically links predicted items on the basis of the value of the essential feature of their selected elements.

Scanning will reduce an active item on the basis of unified lexical material. If the selected element of an active item of form

$$\langle h \leftarrow b_0 \ldots b_n \ ; \ i \ ; ea(i); ea(k) \rangle$$

can be unified with some lexical entry L, and if the reduced lemma rl (i.e., the unified clause minus its selected element) is non-unit, then generate item

$$\langle rl; \ j \ ; ea(j); ea(k) \rangle$$

otherwise, generate item

$$\langle rl; \ \epsilon \ ; ea(k); ea(k) \rangle$$

Two points are important to note here. First, the scanner reduces an active item on the basis of lexical material, which corresponds to some part of the input string or semantics. This means, that for the reduced lemma, this part of the input has actually been consumed. Second, the new generated item inherits the backward pointer of the active item. In principle this means that the scanner defines a lower bound for the chain of predicted items that lead to a unified lexical entry. If the new generated item is an active one, then the selected element initializes a new prediction chain.

Passive completion operates on a passive item and searches for active items, which are in the same item set as the passive one. Thus if the passive item is of form

$$\langle b; \ \epsilon \ ; ea(k); ea(k) \rangle$$

and if there is a unify-able active item of form

$$\langle h \leftarrow b_0 \ldots b_n \ ; \ k \ ; ea(k); ea(l) \rangle$$

and if the reduced lemma rl is active, then generate item

$$\langle rl; \ j \ ; ea(j); ea(l) \rangle$$

otherwise (i.e., rl is passive) generate item

$$\langle rl; \ \epsilon \ ; ea(l); ea(l) \rangle$$

Active completion performs basically the same task with the only difference that it operates on an active item and searches for passive items. Thus if the active item is of form

$$\langle h \leftarrow b_0 \ldots b_n \ ; \ k \ ; ea(k); ea(l) \rangle$$

and if there is a unify-able passive item of form

$$\langle b; \ \epsilon \ ; ea(k); ea(k) \rangle$$

then generate either

$$\langle rl; \ j \ ; ea(j); ea(l) \rangle$$

or

$$\langle rl; \ \epsilon \ ; ea(l); ea(l) \rangle$$

Note that both completion rules can call each other indirectly via ADD-ITEM depending on the form of the generated item.

In summary, the inference rules work together in such a way that the predictor establishes a prediction chain on the basis of the decomposed input, where the completion rules (including the scanner) basically stop a prediction chain, by eventually initializing a new local prediction chain. The next figure 4.12 serves as an illustration of this behaviour.

However, in each case only those items are considered whose essential feature have the same value. Thus each inference rule is only applied on a small subset of items.

4.10 Extending the Un-indexed Version to an Indexed Version

We are now in a position to modify our first version, which we also can call the "un-indexed version" to handle a chart. The agenda control procedure PROCESS need only be changed so that it inspects the initial item set for possible results. For the function MAKE-START-ITEM we have to make sure that the initial state set is created using the value of the essential feature as index. The function ADD-ITEM must be changed in such a way that the item set in which a new item eventually is inserted is chosen from the IN slot of that item. However, this is easy since each item knows the item set it is a member of. The procedure APPLY-TASK can be used unchanged. Thus we have:

$$\langle al; i; ea(i); ea(k) \rangle$$

$$\langle rl_1; i_1; ea(i_1); ea(i) \rangle$$

$$\langle rl_2; i_2; ea(i_2); ea(i_1) \rangle$$

$$\langle rl_{m-1}; i_{m-1}; ea(i_{m-1}); ea(i_{m-2}) \rangle$$

$$\langle rl_m; i_m; ea(i_m); ea(i_{m-2}) \rangle$$

$$\langle rl_{m+1}; i_{m+1}; ea(i_{m+1}); ea(i_m) \rangle$$

$$\langle rl_{m+2}; i_{m+2}; ea(i_{m+2}); ea(i_{m+1}) \rangle$$

Figure 4.12: The relationship of generated items of the different inference rules.

(1)	**proc**	PROCESS(**in:** Goal; **out:** Result):
(2)	StartItem	:= MAKE-START-ITEM(Goal);
(3)	ADD-TASK-TO-AGENDA (StartItem,PRIO(StartItem),Agenda);	
(4)	**while**	NOT(EMPTY-AGENDA-P(Agenda))
(5)	**do**	CurrentTask := GET-HIGHEST-PRIO-TASK(Agenda);
(6)		**if** ADD-ITEM(CurrentTask) **then**
(7)		**do** APPLY-TASK(CurrentTask)
(8)		**if** $\exists$ I $\in$ $S_{\text{VAL}(Goal/Ea)}$; I is of form $\langle \text{ans}; \epsilon; \text{VAL}(Goal/Ea); \text{VAL}(Goal/Ea) \rangle$
(9)		**then** Result := push(I,Result); **fi**
(10)		**od**
(11)	**od**	
(12)	Result :=	**if** empty(Result)
(13)		**then** rejection
(14)		**else** Result **fi;**

```
(1)  proc   add-item(in: Task out Candidate?):
(2)  if     Task is not blocked in S_in(Task) then
(3)  do     ADD Task to S_in(Task);
(4)         Candidates? := true;
(5)  od     else Candidates? := false;
(6)  fi;
```

Only the definitions of the inference rules need some more changes to be sensitive for the state sets of the chart. In principle it is possible that when a new item is generated, then this may eventually cause the creation of a new item set. However, since an item is first added to the agenda these item sets are initially empty.

We now give the specifications of the "indexed" versions of the inference rules. We start with the predictor.

```
(1)  proc          PREDICTION(in: AL; out: Candidates?):
(2)  Candidates?   := false;
(3)  ∀ Rule ∈ G;   Φ = UNIFY(RESCT(SEL(AL)), HEAD(Rule)) and Φ ≠ ⊥
(4)  do            NewItem:= ⟨Φ[Rule]; SF(Φ[Rule], Ea); In; in(AL)⟩
                     with In :=   if unit(Φ[Rule])
(5)                               then in(AL)
                                  else VAL(SEL(NewItem)/Ea) fi
(6)                create S_In if it doesn't exist;
(7)                ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
(8)                Candidates? := true;
(9)  od;
```

Scanning operates on active items and reduces the active items on the basis of found matching lexical material; since this can cause consumption of some portion of the input, the reduced item is eventually added to another item set.

```
(1)   proc          scanning(in: AL; out: Candidates?):
(2)   Candidates?   := false;
(3)   ∀ LE ∈ Lex;   Φ = UNIFY(SEL(AL),HEAD(LE)) and Φ ≠ ⊥
(4)   do            Red := Φ[AL-SEL(AL)];
(5)                 NewItem := ⟨Red; SF(Red,Ea); In; from(AL)⟩;
                      with In :=   if unit(Red)
(6)                                then from(AL)
                                   else VAL(SEL(NewItem)/Ea) fi
(7)                 create S_In if it doesn't exist;
(8)                 ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
(9)                 Candidates? := true;
(10)  od;
```

Passive-completion will be applied on passive items. The FROM slot of the passive item indicates which item set has to be used to look for possible active items. However, the FROM slot will be identical with the IN slot, thus possible active items are in the same item set that the passive item set is a member of.

(1)	**proc**	PASSIVE-COMPLETION (**in**: PL; **out**: Candidates?):
(2)	Candidates?	:= **false**;
(3)	$\forall$ AL $\in S_{from(PL)}$;	Φ = UNIFY(SEL(AL),HEAD(PL)) and $\Phi \neq \perp$
(4)	**do**	Red := Φ[AL-SEL(AL)];
(5)		NewItem:= $\langle$Red; SF(Red,Ea); In; $from(AL)\rangle$
		with In := **if** unit(Red)
(6)		**then** from(AL)
		else VAL(SEL(NewItem)/Ea) **fi**
(7)		create S_{In} if it doesn't exist;
(8)		ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
(9)		Candidates? := **true**;
(10)	**od;**	

Active-completion will be applied to active items. It will search for passive items in its own item set.

(1)	PROC	ACTIVE-COMPLETION (**in**: AL; **out**: Candidates?):
(2)	Candidates?	:= **false**;
(3)	$\forall$ PL $\in S_{in(AL)}$;	Φ = UNIFY(SEL(AL),HEAD(PL)) and $\Phi \neq \perp$
(4)	**do**	Red := Φ[AL-SEL(AL)];
(5)		NewItem := $\langle$Red; SF(Red,Ea); In; $from(AL)\rangle$;
		with In := **if** unit(Red)
(6)		**then** from(AL)
		else VAL(SEL(NewItem)/Ea) **fi**
(7)		create S_{In} if it doesn't exist;
(8)		ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
(9)		Candidates? := **true**;
(10)	**od;**	

4.11 A Parsing and Generation Example

We will use the following grammar fragment to illustrate the behaviour of the new indexed version:[12]

[12] We do not claim that this fragment is linguistically adequate. Its sole function is to illustrate the behaviour of the uniform indexing mechanism.

vp(Sem)←v(Sem) np pp
vp(Sem)←v(Sem) pp np
vp(Sem)←v(Sem) np
np(Sem)←n(Sem)
np(Sem)←np(Sem) pp
pp(Sem)←p(Sem) np

The phrasal backbone of this grammar is context-free. Thus we implicitly assume that strings are represented as difference lists which are simply concatenated. For parsing we can assume that the value of Ea is bound to ⟨PHON D-L⟩ and for generation the value is bound to SEM.

This grammar fragment has the nice property, that for the string "sieht Peter mit Maria" two readings "sehen(peter,mit(maria))" and "sehen(peter(mit(maria))" will be analyzed and for the reading "sehen(peter,mit(maria))" the two strings "sieht Peter mit Maria" and "sieht Maria mit Peter" are generated. Thus the example illustrates very well how we can reuse completed structures in parsing as well as in generation.

The figure 4.13 illustrates the indexing mechanism for parsing the string "sieht Peter mit Maria". We assume that a lemma counter is used that enumerates the lemmas just created (starting from 0) and that the agenda mechanism selects tasks in a depth-first manner. We also count the items that have been placed in some item set starting by 1. The lemma counter will be attached to an item as a prefix, and the item counter as its suffix.

Figure 4.13 graphically outlines the trace through the example. To make things more readable, we are using only the initials of each word of the string. Thus "sPmM" abbreviates the string "sieht Peter mit Maria". The sequence in which item sets are created is indicated by using a counter starting from 0. Thus the index of the initial item set is "sPmM0". The counter will then be used as an abbreviation for the item set indices in an item.

We also show the status of the agenda and the current selected task. We also show those items which represent alternatives but are suspended in an extra row "Item of alternative", to make the depth-first strategy more readable. The trace can be read as follows (using a telegram-like style):

> Add task 0 (the start item) to the initial item set I_{sPmM0} and remove 0 from the agenda. Task 0 is now item 1. 1 predicts two new task, 1 and 2, from which 2 is the current task. Add task 2 as item 2 to I_{sPmM0}. Item 2 scans the word "sieht" and the resulting reduced task 3 is added to the agenda (creating I_{PmM1}); task 3 is added as item 3 to the item set I_{PmM1}; item 3 predicts two new tasks 4 and 5 from which 5 is determined as the current task and added as item 4 to I_{PmM1}; scanning of item 4 fails (the difference lists don't match), so the predictor adds two new

tasks 6 and 7 to the agenda (note that 6 is a recursive call to 4; but this task will be blocked, so that this possible recursive loop is avoided); task 7 is added as item 6 to I_{PmM1}; on item 6 the scanners applies (scanning "peter") and the reduced task 9 is added to the agenda (creating I_{mM2}); task 9 as item 8 predicts task 10; task 10 as item 9 scans "mit" and the resulting task 11 is added (creating I_{M3}); task 11 as item 10 predicts task 12; task 12 as item 11 scans "maria" which creates the passive item 12; item 12 completes task 10 which is added as task 15 to the agenda; task 15 as item 13 completes item 9; the generated task 16 is added as item 14 to I_{PmM1}; item 16 reduces item 2 which causes reduction of the start item 0; the first reading "sieht [Peter mit Maria]" is found; now the second reading "sieht [Peter] [mit Maria]" can benefit from previous precomputation; it also scans "sieht", but can directly be reduced with item 8 and item 15 to yield item 21 and finally item 22; since at this point the agenda is empty, the whole process stops;

Note that the selection function "simulates" a leftmost selection strategy, since in each case it is the leftmost element which essential feature is instantiated. Furthermore, it is important to note that each item set fulfills the equivalence condition as described above, i.e., the inference rules construct well-formed item sets with respect to their definition.

	Agenda	CurrentTask	Item of alternative
22[ans;ϵ;0;0]22			
21[vp;ϵ;0;0]21			
1[vp ←v np pp;0;0;0]18	0	0	1[vp ←v np pp;0;0;0]
18[ans;ϵ;0;0]16	1,2	2	
17[vp;ϵ;0;0]15	1,3	3	
2[vp ←v np;0;0;0]2			
0[ans ←vp;0;0;0]1	1,4,5	5	4[np ←np pp;0;1;1]
$sPmM_0$	1,4	4	
	1,6,7	7	6[np ←np pp;0;1;1]
19[vp ←np pp;0;1;0]19			
16[np;ϵ;1;1]14	1,6,8	8	
8[np;ϵ;1;1]7	1,6,9	9	
7[np ←n;0;1;1]6	1,6,10	10	
4[np ←np pp;0;1;1]5			
5[np ←n;0;1;1]4			
3[vp ←np;0;1;0]3	1,6,11	11	
PmM_1	1,6,12,13	13	12[np ←np pp;0;3;3]
	1,6,12,14	14	
	1,6,12,15	15	
20[vp ←pp;0;2;0]20	1,6,12,16	16	
15[pp;ϵ;2;2]13			
10[pp ←p np;0;2;2]9	1,6,12,17	17	
9[np ←pp;0;2;1]8	1,6,12,18	18	First Result
mM_2	1,6,12	12	
	1,6	6	
12[np ←np pp;0;3;3]17	1	1	
14[np; ϵ;3;3]12	19	19	
13[np ←n;0;3;3]11	20	20	
11[pp ←np;0;3;2]10	21	21	
M_3	22	22	Second Result
	ϵ		

Figure 4.13: A trace through parsing of the string “sieht Peter mit Maria”.

We now demonstrate how the algorithm is used in the generation mode. Note that we need to use the path SEM as essential feature. This is the only requirement to let the indexed version of the uniform algorithm to run for generation in an efficient manner.

Figure 4.14 shows the trace of the semantic expression “sehen(Peter,mit(Maria))”. We make a similar abbreviations for the indices. Thus “s(P,m(M))” abbreviates the semantic expression “sehen(Peter,mit(Maria))”. We also assume that the agenda control processes tasks in a depth-first manner. The item sets are to be read as follows:

> We start by adding task 0 (the start item) to the agenda. Since, it is also the current task, it is added as item 1 into the initial state set $I_{s(P,m(M))0}$; item 1 predicts two new tasks 1 and 2 from which 2 is determined as the current task; task 2 is added as item 2 to state set $I_{s(P,m(M))0}$ which causes scanning of lexical entry “sieht” and the reduced task 3 is added to the agenda (this also creates item set I_{P1}); task 3 as item 3 predicts two new tasks 4 and 5 from which task 5 is added as item 4 to I_{P1}; item 4 scans “peter” and adds new task 6 to the agenda; task 6 as item 5 reduces item 3 which generates task 7 (task 7 creates $I_{m(M)2}$); task 7 is added as item 6 to $I_{m(M)2}$ and predicts new task 8; further processing yields a new item set $I_{m(M)2}$ which results after some more steps task 13 which is added as item 15 to $I_{m(M)2}$; item 13 completes item 7 which creates task 14; task 14 is added as item 12 to $I_{s(p,m(M))0}$ which causes reduction of the start item and the first paraphrase “sieht Peter mit Maria”; the next tasks to process are 10 and 4 (in that order) which do not force creation of new tasks; so the next task is task 1 which is added as item 16 to $I_{s(p,m(M))0}$; after scanning and active completion with item 3 a new task 16 is added to the agenda which is then added as item 17 to $I_{m(M)2}$; item 17 creates a new task 17 which is added as item 18 to I_{P1}; by active completion of item 17 with item 6 the task 18 is created and added as item 19 to $I_{s(p,m(M))0}$; this item leads to the second paraphrase “sieht mit Maria Peter”, and since the agenda is now empty, we stop!

19[ans;ϵ;0;0]20
18[vp;ϵ;0;0]19
1[vp $\leftarrow$v,pp,np;0;0;0]16
15[ans;ϵ;0;0]13
14[vp;ϵ;0;0]12
2[vp $\leftarrow$v,np,pp;0;0;0]2
0[ans $\leftarrow$vp;0;0;0]1
s(P,m(M))$_0$

17[vp $\leftarrow$np;0;1;0]18
4[np $\leftarrow$np,pp;0;1;1]15
6[np;ϵ;1;2]5
5[np $\leftarrow$n;0;1;1]4
3[vp $\leftarrow$np,pp;0;1;0]3
P$_1$

16[vp $\leftarrow$pp,np;0;2;0]17
13[pp;ϵ;2;2]11
8[pp $\leftarrow$p,np;0;2;2]7
7[vp $\leftarrow$pp;0;2;0]6
m(M)$_2$

10[np $\leftarrow$np,pp;0;3;3]14
12[np;ϵ;3;3]10
11[np $\leftarrow$n;0;3;3]9
9[pp $\leftarrow$np;0;3;2]8
M$_3$

Agenda	Current Task	Item of alternative
0	0	
1,2	2	1[vp $\leftarrow$v,pp,np;0;0;0]
1,3	3	
1,4,5	5	4[np $\leftarrow$np,pp;0;1;1]
1,4,6	6	
1,4,7	7	
1,4,8	8	
1,4,9	9	
1,4,10,11	11	10[np $\leftarrow$np,pp;0;3;3]
1,4,10,12	12	
1,4,10,13	13	
1,4,10,14	14	
1,4,10,15	15	First paraphrase
1,4,10	10	
1,4	4	
1	1	
16	16	
17	17	
18	18	
19	19	Second paraphrase
ϵ		

Figure 4.14: A trace through generation of “sehen(Peter,mit(Maria))”

Note that the selection function "simulates" the semantic-head first selection function, although coincidentally in all cases the head element is located in leftmost position. Furthermore, note how the second paraphrase is generated by reusing the PP "mit Maria" (item 13) and the NP "Peter" (item 6) already computed during the generation of the first paraphrase. Since the item sets are indexed by means of semantic information, there is no problem in placing these strings at different string positions as for the first paraphrase. In this example, the item sets are created in sequentially because of the depth-first strategy. If we had used a breadth-first strategy, the item sets I_{P1} and $I_{m(M)2}$ would have been created simultaneously.

In the above simple examples we have abstracted away from most details concerning the grammatical "realism" of this fragment. However, in our implementation we have tested the indexing mechanism not only with the example grammar given in appendix A, but also with an adaption of the English DCG grammar given in [Pereira and Shieber, 1987]. We have also processed constructions that make use of empty elements, such as in the case of *gap-threading* analysis of *topicalization*, *wh-movement*, and *relativization* using this grammar without problems.

However, as known from the work of e.g., [Shieber *et al.*, 1989], [Russell *et al.*, 1990] and [VanNoord, 1993], there is one kind of construction where empty heads are involved which can cause termination problems for any known generation strategy. In the next section we discuss this problem in detail and give our solution to it.

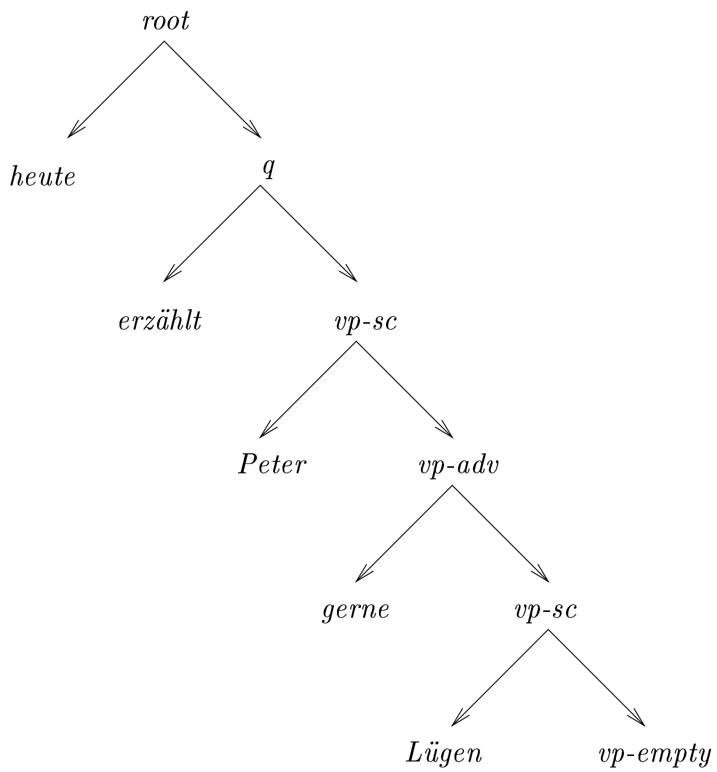
4.12 Processing of Empty Heads

In languages such as Dutch and German, there are two positions in a declarative sentence where tensed verbs may appear: in second position of a main clause, and in final position of a subordinate sentences.

1. "Heute erzählt Peter Lügen"
2. "daß Peter heute Lügen erzählt.'
3. "Heute erzählt Peter gerne Lügen"
4. "daß Peter heute gerne Lügen erzählt"

One possible approach would be to define specific grammar rules to represent the different possible distributions of a verb. However, this causes an unacceptable duplication in the grammar. A more elegant approach would be to use the same verb phrase rule in both subordinate and main clauses and to use a threading mechanism such that the tensed verb in a main clause is "raised" from an "underlying" sentence-final position to a surface second position (see [Netter, 1992] for a more detailed discussion).

In the grammar specified in appendix A (which is an adaption of the one presented in [VanNoord, 1993]) the subcategorization rule (r_1) and the empty vp rule (r_6) realizes such an approach. In these rules the final verb position of a subordinate clause is directly expressed. For main clauses, it is assumed that the tensed verb in second position also occupies the final position, but is phonologically empty. This empty element is defined as the head element of the verbal phrase, which "inherits" the subcategorization and semantic information in the v2 position. Thus, the empty element serves as the head of the verb phrase it is an element of, which means that it will percolate its subcategorization information as well its semantics up to the maximal projection. This means, that the head is involved in a filler-gap dependency. In [Netter, 1992] it is also argued for using binary structures instead of a flat structure, e.g., to allow for insertion of modifiers. This binary branching is expressed in the rules mentioned above and in the modifier vp rule (r_5). Using these rules together yields the following derivation tree for the string "Heute erzählt Peter gerne Lügen":



However, parsing and generation of constructions where such rules are involved is not a trivial task, be they top-down, bottom-up or a mixture of both. And this is true also for our uniform algorithm, if we do not spend some more effort to solve the problem. Solving this problem in the uniform approach causes some problems, because the phenomenon of verb second reveals a real asymmetrical and dual behaviour for parsing and generation.

Parsing For parsing, following a left-to-right on-line strategy, non-termination can occur for subordinate constructions, because the parser will predict subcategorized arguments *before* the verb, that carries the subcategorization frame. If the subcategorization rule (r_1) of the grammar in appendix A would only specify information about phonological information, the predictor has to predict the whole set of grammar rules, which will introduce uninstantiated versions of the empty head rule (r_6) which then causes loops between (r_1) and (r_6). To overcome these problems, we have forbidden prediction of the empty head rule at that position by constraining the value of the feature V2 to be NO.[13] Now, the empty head rule will not be applicable because of the clash of the V2 value.

Generation Consider what happens if we generate from the semantic expression "heute(erzaehlen(peter,luegen))"

If rule (r_3) is chosen, then the selected element will be the verb phrase, because it shares its semantics with the mother node. Using this rule, the predictor will introduce instantiations of the rules (r_1), (r_5) and (r_6). The latter one, however, has no body, so that it can directly be used for completion. Completion with the former introduced rule, will further introduce the same set of rules (modulo restriction), which clearly brings a termination problem.

The basic problem for generation is that the empty head will receive most important information from the filler element, which however, cannot be determined as long as the base case is not found. In order to solve this problem, we have to take into account filler information at the time the empty head rule is predicted.

A first obvious solution would be to redefine the rule that introduces the verb to be argument to the slash value, i.e., to change rule (r_3) such that the verb and the verb phrase share the semantics. Now, our selection function would choose the verb first, whose completion would instantiate the verb phrase proper, so that the above semantic expression can be handled by the algorithm to produce the string "heute erzaehlt peter luegen".

The changed definition expresses that the semantics of the verb second is identical with the whole verbal phrase. In general however, the verb will not be the semantic head of the sentence, at least in the case of adverbial modifiers. Thus using the modification above, it would not be possible to process a sentence like "heute erzaehlt peter gerne luegen'.

But then, we have to live with the original verb second rule. We now present a solution which is similar to the one presented in [Shieber *et al.*, 1990]. Our approach

[13]We assume that only the empty head rule has this feature, and that other empty rules do not.

works as follows. When the predictor is applied on a selected element, that could predict the empty head rule, then *after* unification with the (restricted) selected element but *before* the new item is added to the agenda, we check whether we can scan with the information of the filler (in our grammar, the value of the V2 feature) a lexical entry. Since, the filler has been instantiated with information from the selected element, this will only be possible, if the generator has traversed the semantic expression to the point of finding a verb. For a semantic expression like, "heute(gerne(erzaehlen(peter,luegen)))" this will be the case for the partial expression "erzaehlen(peter,luegen)" but not for "heute(...)" or "gerne(...)". Only, if a matching filler can be found, the thereby "completed" empty head rule will be added to the agenda. This means, that the empty head rule will only be instantiated (including subcategorization information) by genuine potential fillers. The empty head can now serve as the base case for the recursive process in the same way as it would for the lexical filler.

If we assume that the interface between the grammar and the uniform algorithm provides us with a predicate EMPTY-HEAD-P (which returns true if the current grammar rule is the empty-head rule) and function FILLER (which extracts the information from the filler), then the modified prediction rule is as follows:

```
proc            PREDICTION(in: AL; out: Candidates?):
Candidates?     := false;
∀ Rule ∈ G;     Φ = UNIFY(RESCT(SEL(AL)), HEAD(Rule)) and Φ ≠ ⊥
do              if EMPTY-HEAD-P(Φ(Rule)) then
                ∀ Lex ∈ Lex; Ψ = UNIFY(FILLER(Φ(Rule)), Lex) and Ψ ≠ ⊥ do
                NewItem:= ⟨Ψ[Rule]; ϵ; in(AL); in(AL)⟩;
                ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
                Candidates? := true od;
                else do
                NewItem:= ⟨Φ[Rule]; SF(Φ[Rule], Ea); in(AL); in(AL)⟩;
                  with In :=  if unit(Φ[Rule])
                              then in(AL)
                              else VAL(SEL(NewItem)/Ea) fi
                create S_In if it doesn't exist;
                ADD-TASK-TO-AGENDA (NewItem,PRIO(NewItem),Agenda);
                Candidates? := true;
od
```

One could criticise this approach because this solution raises the issue of tuning programs to treat specific problems as they are encountered. Clearly, if an analysis would be available, that is specified in the grammar rules (in the same sense, as we used the V2 feature to cope with empty heads in parsing), the specific modifications are no longer necessary.

In fact, [VanNoord, 1993] proposes a completely different analysis of the verb second phenomenon. In this analysis grammar rules may combine their daughters by more complex string operations, e.g., wrapping operations. However, the problem with van Noord's approach is that parsing cannot be performed on-line, because at least for the head-corner parser Van Noord's approach requires that the whole input string is known in advance. Thus his approach seems problematic for incremental processing. In this case a left-to-right scheduling combines more naturally with an incremental (and on-line) approach.

4.13 Properties

The uniform tabular algorithm is a straightforward extension of the optimized general SLD-resolution rule whose correctness is proven in [Höhfeld and Smolka, 1988]. Thus it also inherits this property. This can be seen as follows.

Each inference rule applies the optimized goal-reduction rule, i.e., it is checked whether two clauses can be combined to build a resolvent, whereby satisfiability of the constraints involved is checked immediately by means of unification. If unification fails, it is known that this part of the search space cannot contain an answer, and consequently the two clauses will not derive a lemma.

The prediction rule generates new items on the basis instantiations of grammar clauses, i.e., it predicts (instantiated) axioms defined by the grammar. The scanning rule performs a reduction of an active item using an instantiation of a lexical clause. Thus it also uses axioms for reduction. The rules passive-completion and active-completion combine two already derived lemmas. However, these completion operations can only be performed if either prediction or scanning has been applied. Therefore, these rules are performed on axioms or consequences of axioms. Since the completion rules reduce the body of an active lemma they will transform it to a passive lemma by repeated application. Thus the selection function has fewer and fewer literals from which to select.

Each inference rule is *fair* because it will generate a set of items where each item correspond to a possible alternative branch at that point of the derivation. Thus, they have a "built-in breadth-first" component, which guarantees completeness.

The indexing mechanism does not destroy this property because of the following reasons. Firstly, the indices are determined directly form the value of the essential feature, which is done after unification successfully takes place. Secondly, the completion rules active-completion and passive-completion need only to consider their corresponding item sets because it is guaranteed that all phrases of a given type are stored in the table before any attempt is made to use the table to look for phrases of that type [Pereira and Shieber, 1987]. Furthermore, passive-completion and active-completion can only take place after prediction and scanning have been performed. Finally, the backward pointer mechanism as defined in 4.9, guarantees that no solution of a sub-goal are out of view because all items in one item set share one common property, namely that they are compatible with respect to the value of the essential feature of one of their literals, which is the head in the case of an unit lemma, and the selected element in the case of an non-unit lemma. Thus, an item set is a *meeting place* of active and passive items, such that an active item looks for some passive item to resolve with, and vice versa, that a passive item looks for an active item which it can resolve. However, both are identical with respect to the value of their essential feature.

All derived candidate items are first added to the agenda before they are placed into the appropriate item sets. The agenda control guarantees that all derived items will be inserted into an item set, which means that they are possibly applied by next application of the inference rules. Before an item is added to an item set, the blocking test checks whether there is already a subsuming one in there. If so, the item is not added. Thus, the blocking test guarantees that only most general lemmas are kept in the item sets.

General flow of control The uniform algorithm is a generalization of the original Early deduction method presented in [Pereira and Warren, 1983] since it can be used for parsing and generation in both a uniform but task-oriented manner. The basic flow of control is a mixed top-down and bottom-up strategy. Prediction is performed top-down by the inference rule prediction. Passive-completion performs a bottom-up completion, where active-completion as well as scanning could be viewed as a kind of top-down completion.

The processing order of the elements of the body of a clause is partially data-driven. Therefore, the grammar itself is an important source of control, since it is the place which defines how a string or semantics expression is to be composed. The selection function we are using is able to dynamically make use of this knowledge. Thus, it depends on the grammar whether the selection is performed in a leftmost or head-driven way or whatever. Moreover, if the grammar also expresses a kind of left-to-right decomposition of semantic information, our algorithm would be leftmost also in the case of generation. On the other hand, if the phonological information has a head-oriented representation, our algorithm would be able to "simulate" a head-corner parser. Thus, although our algorithm is oriented towards efficiency it can handle a wide range of grammar theories.

Top-down versus bottom-up It has often been argued that lexicalized grammars are better to process bottom-up than top-down because the lexicon defines important information for termination and because lexicalized grammars like Categorial Grammar or HPSG only allow very general predictions because of their use of very general rule schemata. Clearly, the first aspect is valid, if only simple top-down methods are used. However, using complex (and clever) strategies, these termination problems are avoided.

If only general rule schemata are used, we can say the following. Because of the general nature of the schematic rules, and because our algorithm tries to look for lexical information as early as possible, the size of a predicted derivation tree is small (in the case of lexical grammars). Only when lexical information is available, its size will increase but be limited with respect to the subcategorization information of the lexical entry. However, if lexical entries are identified for the next parts of the input then more and more constraints for constructing the possible analysis are spread over the derivation tree. But then, more specific predictions are possible, and the algorithm is strongly goal-directed even in the case of lexicalized grammars like HPSG.

Coherence and completeness Coherence and completeness as required by the Ea-proof problem (see section 3.4) is handled in the following way. Note that the Ea-proof problem requires that only and all elements of the input are considered during a proof. The coherence part means that during processing no additional information may be added. To ensure this property, we actually perform scanning in a two step approach. If scanning is tried for some selected element, we first perform a lexical lookup using the first element of the string (i.e., the value of the essential feature) or the current value of the predicate of the lexicon. This is meaningful, since lexical entries are inserted into the lexicon using exactly this information as a key. Thus if lexicon lookup fails the scanning process also fails for this selected element. However, if possible lexical entries can be retrieved we unify them with the constraints of the current selected element. The advantage of performing scanning in this two step way, is that (a) we immediately know of some information in the input which is not valid, (b) that we also can process multiple word entries, and (c) that we only consider information present in the input. Since, it might be the case that some lexical entries do not have a semantic information we require that these entries have a semantic value *nil defining the "null" semantics.

Completeness is checked easily as follows. Note that we require that the reduced start item should be added to the initial item set. However, this is only possible if the value of the essential feature is identical to the input (which has been used to define the index of the initial item set). Thus if we know that the initial item set does not contain an answer we reject the query, even in the case that some other item sets contain an answer.

Termination It is known that unrestricted unification or constraint-based grammars have the formal power of a Turing machine (e.g., [Pereira and Warren, 1983], [Haas, 1989]), thus termination can only be guaranteed for restricted classes of grammar formalisms. For example in section 4.12 we have shown that the subcategorization rule can cause non-termination through completion if we do not restrict the applicability of the empty vp-rule.

From the work of [Bresnan, 1982], [Pereira and Warren, 1983], [Haas, 1989], and [Shieber, 1989] we know that termination can be guaranteed if a grammar is *off-line parsable*, i.e., if it is *finitely ambiguous*. These results, however, have only been worked out for the case of parsing. A termination condition for both parsing and generation is given in [Dymetman *et al.*, 1990] for the class of "Lexical Grammars". It states that if each rule or lexical entry consumes a non-empty part of the input (either string or semantic expression) then termination can be guaranteed.[14] For the case of generation, for example, this means that termination is ensured for grammars in which the lexical entries are defined in such a way, that the semantic structures of each of the elements on the subcategorization list is "smaller" than the semantic structures of the lexical entry itself (see also [VanNoord, 1993]). If a grammar fulfills these criterions our uniform algorithm is guaranteed to terminate.

On-line The current approach is defined as an on-line algorithm in the sense, that it does not initialize the chart for each individual element of the input before these elements are then combined, but rather creates new item sets on demand.

This is especially useful in the case of generation, because in general the generator cannot know in advanced which lexical elements belong to which part of the semantic expression. Of course, our algorithm could easily be adapted to such kind of initialization step, if a process is available, that can assign to each part of the semantic expression a corresponding lexical entry. However, at least in the case of generation, we prefer an on-line approach, because only then syntactic information can be taken appropriately into consideration to restrict lexical access during early stages of processing.

Furthermore, using an on-line approach together with top-down prediction is necessary to satisfy the *valid prefix property* as discussed for example in [Schabes, 1990] and it is also an important basis for *incremental processing* which we will consider in more detail in the final chapter of this thesis.

Uniform indexing mechanism The indexing mechanism that we have introduced could be seen as a generalization of the Chart or tabular approaches developed in the area of parsing algorithms but now also available for generation. Since we use the value of the essential feature, the indices of all item sets together simulate a kind of input traversal. For example, if the grammar describes a left-to-right concatenation of strings, then the indices describe the left-to-right traversal of the string, so that the index of a "right" item set is a proper substring of the index of a "left" item set. In the case of generation, the indices can be viewed as being constructed by a means of predicate/argument traversal. Since, we view a predicate/argument construction structurally as a tree, generation realizes a kind of tree traversal.

[14] A formal treatment of this property can be found in [Dymetman, 1994].

Agenda-based control The inference rules have been embedded in a flexible agenda mechanism. This allows us to realize not only depth-first or breadth-first strategies but also more sophisticated preference-based strategies. Although we have only used a randomized priority function as a third alternative to depth-first and breadth-first, it should be clear that if the grammar assigns preferences to its elements, we can easily take advantage of these preferences. Thus, our uniform algorithm is also open to such preference-based strategies.

Using an agenda mechanisms allows us to suspend the addition of an item to its item set until this item is chosen as the current task of the agenda. Each assigned priority of an item can be viewed as a kind of time stamp, which specifies when the item should be considered for processing. If the priority queue is handled as a stack realising a kind of depth-first or best-first strategy and if we suspend processing after the first successful result has been found, then only those items are added to the chart, that were needed for deriving this result. Thus only a subset of generated items is considered by the inference rules as well as by the blocking rule.

4.14 Implementation

The uniform tabular algorithm is fully implemented in Common Lisp. It uses two well-tested existing tools: $\mathcal{UDiNe}$ an advanced feature constraint-solver [Backofen and Weyers, 1993] and Fegramed, an interactive graphics editor for feature structures [Kiefer and Fettig, 1994]. Both tools have been developed as parts of the DISCO system, a natural language core system for German [Uszkoreit *et al.*, 1994].

$\mathcal{UDiNe}$ is a feature constraint solver capable of dealing with distributed disjunctions over arbitrary structures, negative co-references, full negation, and functional and relational constraints. It is the first (and to our knowledge the only) implemented feature constraint solver that *integrates* both full negation and distributed disjunctions. Although we have not made use of disjunction in this thesis, we have made use of them in some small grammars in order to show that our algorithm is also capable in dealing with more complex feature representations.

We will now describe some of the implementational aspects of our uniform tabular algorithm. This implementation description is not meant to be the only way or the best way to implement our uniform algorithm.

Representation of grammar and lexical entries $\mathcal{UDiNe}$ is used to represent feature structures. Feature structures are represented in a string-like notation where variables are prefixed with the sign %. Such string notation is then transformed by a special $\mathcal{UDiNe}$ reader in some internal Lisp structures.

Definite clause specifications are then described in Lisp-list notation, where a definite clause is represented as a list, where the first element represents the head and the remaining elements represent the body of a clause. Each relation is represented as a list, where the first element represents the relational atom and the remaining elements represents the arguments of the relations directly specified as feature structures in string notation. The following structure shows the Lisp representation of rule (r_1):

```
((sign "[(cat vp)
         (sc %Tail)
         (sem %Sem)
         (lex no)
         (v2 %V)
         (string [(d-l %p0)
                  (e-l %p)])
         (deriv [(name vp-sc)
                 (dtrs [(first %1)
                        (rest [(first %2)
                               (rest end)])])])]")

     (sign "%Arg=[(sc end)
                  (v2 no)
                  (string [(d-l %p0)
                           (e-l %p1)])
                  (deriv %1)]")

     (sign "[(cat vp)
             (sc [(first %Arg)
                  (rest %Tail)])
             (sem %Sem)
             (v2 %V)
             (string [(d-l %p1)
                      (e-l %p)])
             (deriv %2)]"))
```

Grammar and lexical rules are transformed in some internal representation, where the grammar and lexicon are stored as a hash table whose key values are lists of possible entries. For grammar rules the relation name of the head of a clause and its arity is used to construct a symbol which is used as the key in the hash table. Thus, the above rule would be inserted under the key SIGN/1. However, storing of grammar rules in a hash table in that way only makes sense if we use a set of different relational symbols. For the grammar in appendix A however we only used the relation name SIGN. In order to retrieve grammar rules efficiently, we therefore use the value of the feature CAT for determining the hash key. The effect is that all grammar rules with same category value are stored in a list under the same key. Our implementation can be parameterized with respect to this issue.

For lexical rules we actually use two different hash tables, where both point to the same set of lexical entries (i.e., the same set of entries can be retrieved from two different directions). For parsing we use the first string element (which can be accessed via the path ⟨STRING D-L FIRST⟩, and for the generation we are using the predicate name (via the path ⟨SEM PRED⟩).

Using these data structures, scanning is performed in two steps. First, for the selected element of an active item, a possible key is determined which is either found under the path ⟨STRING D-L FIRST⟩ or ⟨SEM PRED⟩. If such a key can be determined the corresponding entries in the lexicon are retrieved and a list of possible candidates are returned (which can be the empty list, if no entries exist for this current key). Each candidate is then unified with the constraints of the selected element. If the constraints of the selected element do not specify any information about a string or a semantic information, no key can be determined. If this happens, the scanner does not apply. This is also the case if no lexical entry for a key can be determined.

In order to syntactically distinguish between definite clauses, that represent grammar rules and lexical rules, we prefix each clause either with the special grammar rule symbol "$< -$" or with the lexical entry symbol "$<< -$". Thus the above grammar rule would be rewritten as

(< − (sign "...") (sign '...") (sign "..."))

where as a lexical entry would be written as

(<< − (sign "..."))

Both symbols are actually macros that call the respective functions.

Representation of item sets The structure of an item is represented as a record structure using the defstruct construct of Common Lisp in the following way:

```
(defstruct item
  clause           ;; internal representation of a definite clause
  selected-element ;; the selected element
  position         ;; its position
  unit             ;; T if clause is unit otherwise NIL
  in-set           ;; the index of the item set item is a member of
  from-set         ;; the backward pointer
  number           ;; the number of the item
  ignore           ;; indicates whether the item should be ignored or not
   )
```

The inference rules only use the value of the slot selected-element. We explicitly represent the position for efficient reduction of a clause.

The structure of an item set is a record of the following form:

```
(defstruct item-set
  index          ;; the index of the item set
  active-items   ;; a hash table that keeps active items
```

```
  passive-items  ;; a hash table that keeps passive items
  number)        ;; the number of the item set
```

A global list is then used to store the several item sets.

The selection function The selection function returns two values: the selected element and its position. The Lisp definition of the selection function currently used is as follows

```
(defun dynamic-selection (clause &aux (body (clause-body clause)))
  (if body
     ;; if clause has a body
      (multiple-value-bind (selected-element pos)
          ;; determine the selected-element and its position
          (get-next-dtr body)
        (if selected-element
           ;; return both if found
            (values selected-element pos)
           ;; otherwise choose the leftmost, which has pos 0
          (values (first body) 0)))
    ;; NIL is used to indicate epsilon
    (values nil nil)))

(defun get-next-dtr (body)
 "loop through the elements of the body until there is
a body whose essential feature is instantiated. In that
case return the relation and its position. Otherwise return
NIL."
  (let ((cnt 0))
    (dolist (relation body (values nil nil))
      (if (if (equal *ef* *sem-path*)
               (get-sem-constraints relation *sem-path*)
             (get-string-constraints relation *phon-path*))
          (return (values relation cnt))
        (incf cnt)))))
```

We use a dispatching mechanism to store the selection function to be used by the inference rules. This is easily done in Lisp by storing the selection function as a property to a specific Lisp symbol. The inference rules then have to access this property field and then call the bound function to the current arguments. In our system we use the symbol :selector and the property :function to bind the selection function. Thus, when a new item is created, the currently bound selection function is activated by the call:

```
(funcall (get :selector :function) clause)
```

We have used such a mechanism to be able to experiment with several selection functions without the need for recompiling the whole code. Furthermore, if it is known that for a specific grammar the leftmost selection function is needed, it is very easy to define such a function and to make it available to the inference rules. For example the definition of the leftmost selection function could be:

```
(defun leftmost-selection (clause &aux (body (clause-body clause)))
   (if body
       (values (first body) 0)
     (values nil nil)))
```

Thus, our implementation is modular with respect to the selection function.

Inference rules The Lisp definitions of the inference rules are all very similar and more or less follow the abstract definitions as used in the previous sections. The schematic structure of an inference rule is as follows:

```
(defun <inference-rule> (item)
  (let* (<x>
         ;; X := either the selected-element or the clause of the item itself
         <candidates>
         ;; possible candidates
         (coob (make-control-obj *global-fail-context-set*))
         (candidates? nil))

    (if (consp candidates)
        (dolist (candidate candidates candidates?)
          (let* ((unify-res (unify-clause x candidate coob))) ; call unifier
            (unless (eq unify-res fail) ;; if MGU exist
              (let ((new-item
                      (make-item candidate <some-more>)))
                   (add-task-to-agenda new-item
                                      (compute-priority new-item
                                       :producer <something>)
                                        *agenda*)
                    (setq candidates? T)))
            (reset-ctrl-obj coob))))))
```

The set of possible candidates are determined in dependence of the status of the current item (active or passive) and of the specific task the inference rule is defined to fulfill. The unifier $\mathcal{UD}i\mathcal{N}e$ performs unification destructively, keeping a set of control objects which can be used to "reset" the effects of destructive unification. On each candidate the unifier is applied. If unification does not fail, a corresponding new item is created and added to agenda *agenda*. In our current implementation, each clause of a new item is copied. Thus we only have to keep one control object which has to be reseted after unification takes place (in all cases).

Note that because of the general scheme for inference rules, it might also be possible to define additional inference rules, for example one, that performs some sort of bottom-up prediction. We have actually implemented such rules (for experimental reasons), however, one should be aware of the fact, that this might change the characteristic behaviour of the whole algorithm. For this reason, we do not want to discuss this further possibility.

The agenda control The Lisp definition of the basic agenda control looks as follows (abbreviated where convenient):

```
(defun prove (goal)
  (let ((goal-pred (predicate (clause-head goal)))
        (start-item (make-start-item goal)))

    (add-task-to-agenda start-item
                        (compute-priority)
                        *agenda*)


    (do ((task (get-highest-priority-task *agenda*)
               (get-highest-priority-task *agenda*)))

        ((null task) (if (consp *result*) T))

    (if (add-item task) ;; only adds item if not blocked;
      (let ((result (get-answer goal-pred task)))
           ;; check initial item set
         (if result
             (progn
               (setf (item-ignore result) T)
             ;; ``simulates'' removal of found answer from item set
               (push result *result*)
               (if *all-p* ;; non-interactive mode
                   (process-one task)
                 (progn
                   (print-result (lemma-clause result))
                   ;; used fegramed to show result
                   (if (yes-or-no-p
```

```
                         "~&Should we continue looking for solutions?  ")
                        ;; Prolog-like interactive mode
                        (process-one task)
                      (return T)))))
           ;; if currently no answer has been found
           (process-one task)))))))


(defun process-one (item)
   (if (item-unit item)
      (passive-completion item)
    (or (active-completion item))
       (progn
         (prediction item)
         (scanning item))))
```

4.15 Item Sharing Between Parsing and Generation

The uniform tabular algorithm developed so far is new and exhibits several relevant novel ideas, e.g., a dynamic selection function for parsing and generation and the use of a uniform indexing mechanism for both tasks. It combines previous approaches for parsing and generation in one computational framework, and shows that parsing and generation can be realized uniformly and efficiently. Since we have considered parsing and generation in an isolated way, we are still in the main stream of grammatical processing.

However, we now present a novel method for grammatical processing, namely the use of items produced in one direction (e.g., parsing) directly in the other direction (e.g., generation). We will call this method *item sharing* between parsing and generation.

If one assumes that parsing and generation are performed in an isolated way, then such method seems to be an overhead. However, in the next chapter we will argue at length that a tight integration of parsing and generation is necessary in order to handle performance aspects like monitoring, revision or generation of paraphrases. On the basis of this discussion we present a set of new methods (in order of increasing complexity) that benefit from the uniform tabular algorithm in combination with the item sharing approach.

To my knowledge, the possibility of sharing items between parsing and generation has not been addressed in the literature so far. Under this perspective, our item sharing approach is a new dimension in the area of uniform processing of reversible grammars and demonstrates that the integration of parsing and generation can be achieved in an efficient way.

4.15.1 The Basic Idea

In this subsection I will outline the realization of the item sharing method. Assume that the uniform algorithm is in the parsing mode. Then in each case a passive item is computed we automatically make available this item also for the generation mode. Thus, for example, if we are going to generate from the semantics of the parsed input we directly can return the previously computed answer during parsing as result of the generation mode (i.e., if we only consider one paraphrase). More over, if we perform generation using a different semantics as the "parsed" one, but which is identical with respect to some partial semantics structures (e.g., some arguments are semantically identical), then the generator also can "reuse" the results determined through parsing. Clearly, this kind of processing makes only sense if during parsing and generation the same grammar is used.

The restriction of sharing only passive items is plausible for the following reasons. Note that passive items have no selected element, and the value of the IN and FROM slots are the same. Assume we are in the parsing mode. Then, by means of the definition of item sets (see section 4.9), the appropriate values for the IN and FROM slot for the direction mode can directly be determined on the basis of the semantics information of the passive item. This guarantees that shared passive items produced during the parsing mode, are at their right places when they are used by the generation mode.

On the other hand, for active items, in general the chosen selected elements during parsing and generation will be different, and the essential argument of the other direction will be un-instantiated. Therefore, it would not be possible to determine the right place of an shared active item as it is the case for shared passive items.

On the basis of these observations, the structure of an item sharing approach using the uniform tabular algorithm is as follows: We assume that the uniform algorithm maintains two different agendas, one for the parsing mode and one for generation. This is no overhead, because it allows us to order the tasks of an agenda using, for instance, different preferences. Since items sets are considered as equivalence classes, that are determined on the basis of the value of the essential argument, we assume that parsing and generation have different item sets. Item sets consist of active and passive items. Now, we require that passive items are shared between the item sets determined during parsing and generation. This means, that the parser and generator each have there own private active items but can operate on the same set of passive items. Figure 4.15 illustrates the structure of the item sharing approach.

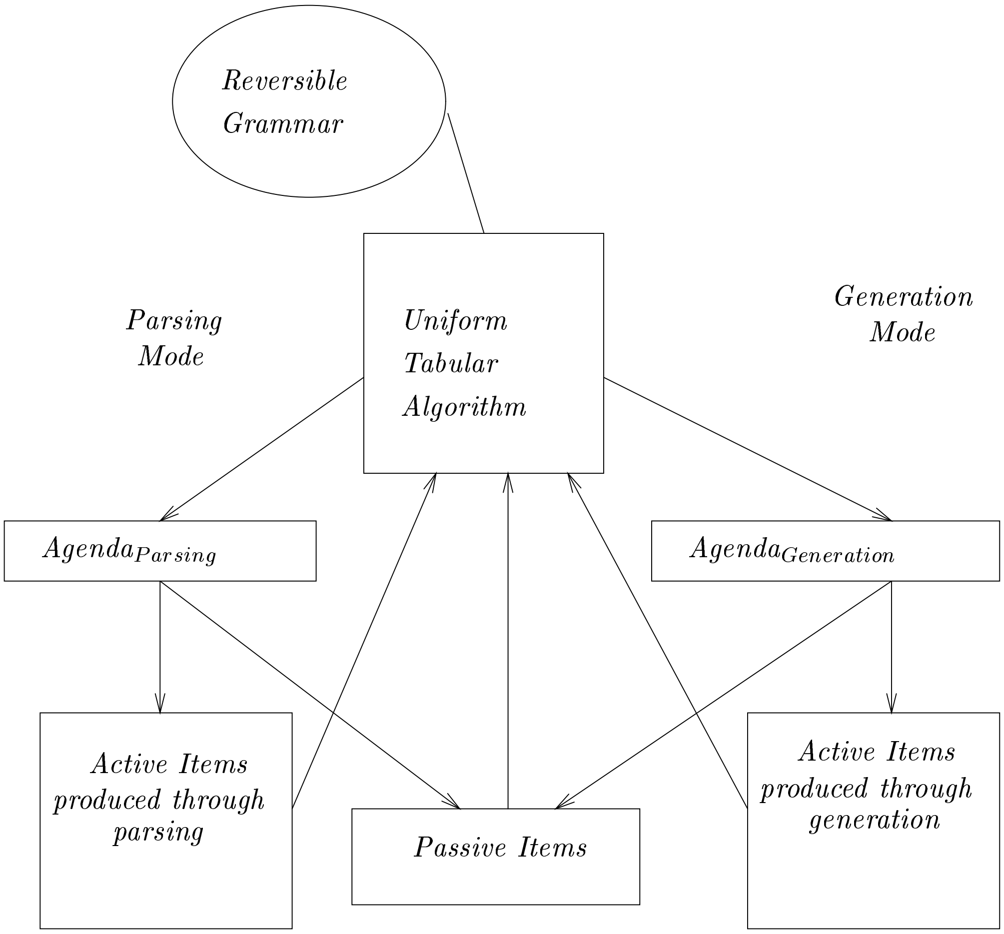


Figure 4.15: The *item sharing* approach using the uniform tabular algorithm. During the different modes the uniform algorithm maintains different agendas and private active items for the different modes. However, passive items are shared by both directions.

4.15.2 Adaptation of the Uniform Tabular Algorithm

In order to adapt the uniform algorithm for the item sharing method we change the structure of an item such that it contains different slots IN and FROM for parsing and generation. Thus, we have

$$\langle L; i; in_p; from_p; in_g; from_g \rangle$$

where L denotes the lemma of an item, i the (position of the) selected element. During parsing the slots IN_p and FROM_p are used and during generation the slots IN_g and FROM_g.

If we are in one of the two possible directions, say parsing, then for active items only the corresponding IN and FROM slots are filled with values of the essential feature. The other slots are unbounded which will be denoted by using the symbol NONE. We will use the notation PHON(I) to denote the value of the essential argument used during parsing and SEM(I) to denote the value of the essential argument used during generation. Then the general structure of active and passive items is as follows. In the case of parsing, active items are of form

$$\langle al; i; phon(i); phon(k); none; none \rangle$$

and for passive items we have

$$\langle pl; \epsilon; phon(m); phon(m); sem(m); sem(m) \rangle$$

where al is an active lemma with selected element at position i in the body of al, $phon(i)$ is the index of the item al is a member, and $phon(k)$ is the backward pointer. pl is a passive item with no selected element, and m the pointer to the head of the passive lemma. Note that in the case of passive items, the value of the essential arguments for both parsing and generation are determined on the basis of the constraints of the passive lemma's head. This is consistent with respect to the definition of item sets.

Analogously, for generation active items are of the form

$$\langle al; i; none; none; sem(i); sem(k) \rangle$$

and for passive items we have

$$\langle pl; \epsilon; phon(m); phon(m); sem(m); sem(m) \rangle$$

Now the inference rules can easily be adapted to handle such item structure. Firstly, the uniform algorithm only considers one IN-FROM pair, depending on the major mode, for example parsing. If a new re-solved lemma (determined through prediction or completion) is active, only the IN-FROM pair of the major mode is filled with the value of the essential argument. The value of the generation slots are by default NONE. However, if a passive lemma is re-solved the slots used by generation receive values determined on the basis of the essential feature specified for this direction (i.e., the value of the SEM path). However, this is actually only done, if the essential argument is indeed instantiated, otherwise the default value NONE is used. This guarantees that for passive items, where only the essential feature of one direction (for parsing this is the string feature) is instantiated are not shared, which is necessary, if empty rules are involved. As a consequence, we forbid that empty elements (like gaps) are not shared unless they are filled with some value.

4.15.3 An Object-Oriented Architecture

Clearly, to be able to assign the correct values to the IN and FROM slots, the uniform algorithm has to know in which mode it is. To make this an automatic task the uniform algorithm has been embedded in an *object-oriented* environment. In this environment parsing and generation are defined as instances of a class PROOF, and the control mechanism of the underlying object-oriented language automatically will choose the right slots.

The item sharing method has been implemented on the basis of the implementation of the uniform tabular algorithm. Since Common Lisp now also entails the Common Lisp Object System (CLOS for short, [Steele, 1990]) as an standard extension, it has been very easy to embed the uniform tabular algorithm in an object-oriented environment.[15] The definition of the general class PROOF is as follows:

```
(defclass proof ()
  ((name :accessor name :initarg :name
         :documentation "The name of the prover.")

   (sf :accessor sf :initarg :sf
       :documentation "The specified selection function.")

   (start-item :accessor start-item :initarg :start-item
               :documentation
                 "The item that carries the goal to prove.")

   (result :accessor result :initarg :result
           :documentation
```

[15] In [Keene, 1989] and [Winston and Horn, 1989] good introductions to CLOS can be found. In [Neumann, 1993] we describe an CLOS-based framework for natural language system design. This approach has been used for the realization of the architectural platform of the systems DISCO and COSMA, cf. [Uszkoreit *et al.*, 1994].

```
            "Holds the result of the proof.")

   (agenda :accessor agenda :initarg :agenda
          :documentation
             "The agenda processed by the prover.")

   (task-counter :accessor task-counter :initarg :task-counter
                :documentation "Counts tasks from 0 upward.")

   (prio-fct :accessor prio-fct :initarg :prio-fct
            :documentation
              "The priority function to be used by the agenda.")

   (restrictor-fct :accessor restrictor-fct :initarg :restrictor-fct
                  :documentation
                    "The restrictor function defined for prover instance.")

   (lex-access-fct :accessor lex-access-fct :initarg :lex-access-fct
              :documentation
                "The function that performs lexical access.")
   )
  )
```

Note that for the above definition as well as for the definitions that come we make use of implementational aspects already described in 4.14.

Parsing and generation are simply defined as subclasses of this class and instances are created in the following way:

```
(make-instance 'parse
      :name "Parsing direction"
      :sf #'dynamic-selection
      :agenda (make-agenda)
      :task-counter 0
      :prio-fct #'depth-first
      :restrictor-fct #'restrict-clause
      :lex-access-fct #'string-access
      )


(make-instance 'generate
      :name "Generation direction"
```

```
:sf #'dynamic-selection
:agenda (make-agenda)
:task-counter 0
:prio-fct #'depth-first
:restrictor-fct #'restrict-clause
:lex-access-fct #'sem-access
)
```

Note that the slot :lex-access-fct is the only slot that is assigned with different (functional) values.

All functions (with the few exceptions given below) are defined as methods for the class PROOF. This means, that the control function, the inference rules, as well as the agenda mechanism is still the same, and defined only once. In some sense, this means that the uniform tabular algorithm only exists one time, but is used by the two different instances. The advantage of using two different instances is that we can easily maintain different agendas or can use specific priority functions for both instances. Thus, our implementation directly mirrors the architecture of the item sharing approach as shown in figure 4.15.

The only functions that are defined as specific methods for the parsing and generation classes are MAKE-ITEM and ADD-ITEM (see section 4.14), and they differ only with respect to one additional call of a function. For the case of MAKE-ITEM, we have to provide, that if a new lemma is passive, we have to determine values for the slots of the direction that is currently not active. And for the case of ADD-ITEM we additionally have to add the new item to the corresponding item set (which has been created trough MAKE-ITEM, if the new lemma is passive) maintained by the inactive instance. Note that this does not mean that the new item is copied, but that the parsing and generation instances actually share this item (internally this means that Lisp provides two pointers to the same internal object).

To illustrate the behaviour of the item sharing approach consider the parsing and generation example given in section 4.11 (see also figures 4.13 and 4.14). For example, when during parsing the passive item for the partial string "mit Maria" is re-solved (in figure 4.13 it is the item with task counter 15 and item number 13), then this will cause the creation of an item set for generation with index "mit(Maria)". Additionally a pointer to the passive item 13 is established. In the item sharing approach the structure of the item 13 is (making use of the abbreviations introduced in section 4.11):

$$15[pp; \epsilon; `mM'; `mM'; `m(M)'; `m(M)']13$$

Thus if we perform generation with the semantics "sehen(Peter,mit(Maria))" the "parsed" passive item for "mit Maria" with semantics "mit(Maria)" can directly be used during the generation mode.

4.15.4 Implementation

The item sharing approach has been implemented as an object-oriented extension of the uniform tabular algorithm as described above. We have further provided some parameterization with respect to possible applications. For example, it is possible to specify whether during one direction shared items should be ignored or not. This means that only during one direction, say parsing, passive items computed during generation are used, but that generation will ignore passive items computed during parsing. In the final chapter of this thesis we give an example of an application which can make use of this kind of parameterization.

Furthermore, also in the case of the shared item method the Prolog-like interactive mode is still available (see section 4.14). This gives the possibility to simulate some sort of preference-based behaviour, because the user can interactively choose the most appropriate answer. In the final chapter we will come back to the issue of preference-based strategies.

In the chapter 5 we show in detail how the item sharing approach is used for performing self-monitoring and revision during generation.

4.16 Conclusion

In this chapter we have presented a uniform tabular algorithm for parsing and generation of constraint-based grammars and a method for sharing items between both processing directions.

The uniform tabular algorithm is based on Pereira and Warren's Earley deduction method. However, for obtaining an efficient and task-oriented behaviour of the uniform algorithm we use a data-driven selection function and a uniform indexing mechanism for efficient retrieval of partial results.

Both the selection function and the indexing mechanism are parameterized with respect to the input in question (a string for parsing and a semantics expression for generation). Since the only relevant parameter our algorithm has with respect to parsing and generation is the difference in input structures the basic differences between parsing and generation are simply the different input structures. This seems to be trivial, however our approach is the first uniform algorithm that is able to adapt itself dynamically to the data, achieving *a maximal degree of uniformity for parsing and generation under a task-oriented view.*

Besides the facts that the new uniform tabular algorithm is data- and goal-directed and maintains partial results in a practical way, the uniformity of the indexing mechanism also makes possible that parsing and generation are able to share their results. We have – for the first time – presented such a technique, which we call *item sharing.* This method is in particular useful and important if parsing and generation have to be interleaved in solving specific problems like paraphrasing or revision. The item sharing approach developed in this thesis shows for the first time how a tight integration of parsing and generation can be done efficiently and effectively. In the next chapter we are going to explain in detail how the uniform tabular algorithm in combination with the item sharing approach is used to realize such a behaviour.

Chapter 5

A Performance Model based on Uniform Processing

The uniform tabular algorithm just developed together with a reversible grammar constitutes the grammatical competence base of a natural language system. The grammar declaratively describes the set of all possible grammatical well-formed structures of a language and the uniform algorithm is able to find all possible grammatical structures for a given input – at least potentially.

Grammatical knowledge and processing in this sense is necessary to render possible natural language processing. This is not only a principal emphasis in computational theories of human language ability but also in artificial intelligence [Wahlster, 1986]. However, if we accept the view that the primary motivation for developing natural language systems is to facilitate *human-machine* communication and/or to extend, clarify or verify theories that have been put forth in linguistics or cognitive psychology then it is necessary to investigate the relationship between linguistic competence and linguistic performance. This means that we have to consider the grammatical competence under the perspective of language use, for example, in order to explain how preferences, disambiguation, paraphrasing, incrementallity, monitoring and revision or active vs. passive language use can be explained on the basis of language competence.

The scientific goal followed in this chapter is to explain how such methods can be modelled by means of the uniform grammatical competence base. Of course, it would exceed the scope of this work to consider all of these aspects. Therefore, we concentrate ourself on the issue of monitoring, revision and paraphrasing during natural language generation. In the final chapter we outline how the uniform grammatical model can also be used to accommodate other aspects, like the use of preferences and fully incremental text processing.

It is a fact that speakers monitor what they are saying and how they are saying it [Levelt, 1989]. When they are making a mistake, or express something in a less felicitous way, they are able to make a repair. It is furthermore evident that speakers monitor almost any aspect of their speech, ranging from content to syntax to the choice of words to properties of phonological form and articulation (see [Levelt, 1989], page 497). In this thesis we are concerned with monitoring on the grammatical level and explain how revision can be performed in order to reduce the risk of generating ambiguous sentences that could be misunderstood by the listener. However, the methods we are going to develop can also be used to handle other monitoring and revision strategies. We demonstrate this by showing how the new methods are used during natural language understanding as a means of disambiguation. Furthermore, in the final chapter we outline how these methods in combination with preference strategies can be used to realize hearer adaptable strategies.

The underlying strategy followed in the methods to be presented is a tight interleaving of parsing and generation. It will turn out that the uniform tabular algorithm in combination with the item sharing approach leads to an elegant and effective specification of these performance-oriented methods. Furthermore, the methods can be straightforwardly be combined with the uniform algorithm in such a way that the original behaviour of the uniform algorithm is not affected. This means that it is possible to interrupt the monitoring and revision process in such a way that further processing proceeds in the normal un-monitored way. Thus, we achieve an elegant synergy of competence and performance-oriented processing.

Overview

This chapter is organized as follows. In the next two sections we discuss the modular status of natural language systems and the consequences of grammar reversibility for the system's design. We will show that the use of a uniform grammatical component leads to more compact systems and helps to identify a clean linguistic and conceptual separation. On the other hand, this kind of modularity implies a serious problem, namely that the conceptual component cannot control completely whether and how the linguistic system will realize a given semantic structure.

In section 5.3 we will argue that in order to maintain a modular design additional mechanisms are necessary to perform some *monitoring* and *revision* of the generator's output. We will argue that the best way in realizing these mechanisms is by means of a tight integration of parsing and generation. Before, however, in section 5.4 the new competence-based performance model is presented we discuss in section 5.3 previous approaches that also take an integrated view of parsing and generation.

In order to make clear how the new model is used to solve particular problems, we present in section 5.5 a fundamental generation strategy, namely that of not producing ambiguous output. The idea here is that the generator has to run its output back through the understanding system to make sure it's unambiguous. In particular we demonstrate how reversible grammars are used to make such strategy effective and efficient.

In section 5.6 we also apply this method during the understanding mode of a NLS for the purpose of disambiguation by means of the generation of paraphrases. The idea here is that after parsing an utterance, then if this utterance has several readings, corresponding paraphrases are generated, that reflect the semantic differences. The user is then asked to choose the one he intended.

An important property of both methods is that the nature of the underlying *parsing* and *generation* strategy is not important, i.e. the strategy can be used with any parsing- or generation strategy. However, an obvious restriction for both is that monitoring only takes place when the generator has finished computing a first string. Therefore, the underlying monitoring strategy could also be denoted as a *non-incremental generate-parse-revise* strategy.

The basic strategies followed in the non-incremental approach also allow the specification of a monitoring strategy that interleaves generation and parsing more tightly in such a way that monitoring can take place even *during* generation. In section 5.7 we present such an approach. As it will be clear such an *incremental approach* is much more flexible and natural. The method we are presenting takes full advantage of the uniform algorithm. In particular, the top/down generation approach followed in the uniform algorithm as well as the item sharing approach makes the incremental monitoring and revision method a practical one.

5.1 The Modular Status of Natural Language Systems

Natural language processing is often viewed as a *complex modular system* consisting of interacting components whose mode of operation may be radically different. The different components can be collected into two subsystems according to their different tasks:

- a linguistic system
- a conceptual system

In the case of language understanding, the basic task of the linguistic system is to determine a semantic representation of a given utterance from which the conceptual system can draw general inferences, e.g., in order to resolve anaphora, and to determine the speakers intention behind the utterance. In order to perform these tasks the conceptual component acts primarily on the basis of world knowledge, discourse model and situation knowledge. The basic knowledge sources of the linguistic system are the grammar and the lexicon of a language which represents in a declarative way the relation between well-formed utterances and their associated semantic representations. The central process for analysing the grammatical structure of a given utterance is called parsing. The output the parser delivers is the set of possibly all semantic representations that the grammar associates to that utterance.

In most natural language understanding systems NLU, i.e. systems where only language understanding is considered, a *clean separation* between the two systems is assumed. In this view, the linguistic system is modular in the sense, that it is the only component that is concerned directly with the form and content of the grammar while the conceptual system is the only one that is responsible for general inference. In these kind of models the semantic representation specified in the grammar serves as an *intermediate representation* between the linguistic and conceptual system.

This division of labour is also the basis of current natural language generation systems NLG developed in the area of artificial intelligence and computational linguistics (see e.g.,[Dale *et al.*, 1990; Paris *et al.*, 1991] for a collection of state-of-the-art reports) as well as in cognitive science (see e.g., [Levelt, 1989]). It is an increasing consensus that the input to an NLG should be of the form of a COMMUNICATIVE INTENTION, i.e. some goal that the speaker wants to communicate by means of natural language. To be able to produce an utterance that adequately communicates the speaker's goal several subtasks have to be performed, e.g.,

- the determination of the content of an utterance
- the organisation of that content in a coherent discourse
- the determination of its linguistic realization

Most of the work of determining the content of a discourse is done by the conceptual component. In order to perform its tasks it also takes into account world knowledge, discourse and dialog knowledge as well as knowledge about the interlocutors (however, not necessarily represented in the same way or the same account of knowledge). The linguistic system is responsible for realizing the content of a discourse determined by the conceptual component in a natural language. On the basis of a grammar and a lexicon the grammatical structure of a given content has to be produced in order to determine a well-formed utterance. The process that is responsible for this task will be called *grammatical generation* or short *generation.*[1]

Although the modular design of an NLG into a conceptual and linguistic component has been proven fruitful for the investigation of natural language production it is a matter of active debate what the input for the linguistic component exactly should look like. For example, many researchers (e.g., [Danlos, 1987; Appelt, 1985; Hovy, 1987; Finkler and Neumann, 1989; Reithinger, 1991; Neumann, 1991a]) have argued that the conceptual and linguistic decisions are strongly dependent upon each other; e.g., in the case of lexical gaps, choice between near synonymous or paraphrases a communication between the two phases is required. On the other

[1] Thus viewed, we ignore for the moment relevant aspects of generation, like morphological and speech generation, since the main point to discuss already falls out without explicit consideration of these aspects.

hand in many approaches it is assumed that the conceptual component has to provide all information needed by the linguistic component to make decisions about lexical and syntactic choices (e.g., [McDonald, 1983], [McKeown, 1985], [Levelt, 1989], [McKeown *et al.*, 1990]). They assume that feedback from the linguistic to the conceptual component would be exception rather than rule. The different views have lead to different architectures where the line between the conceptual and linguistic component have been drawn in different ways, i.e. there is no such clean separation between the two components as it is the case for most NLU systems.

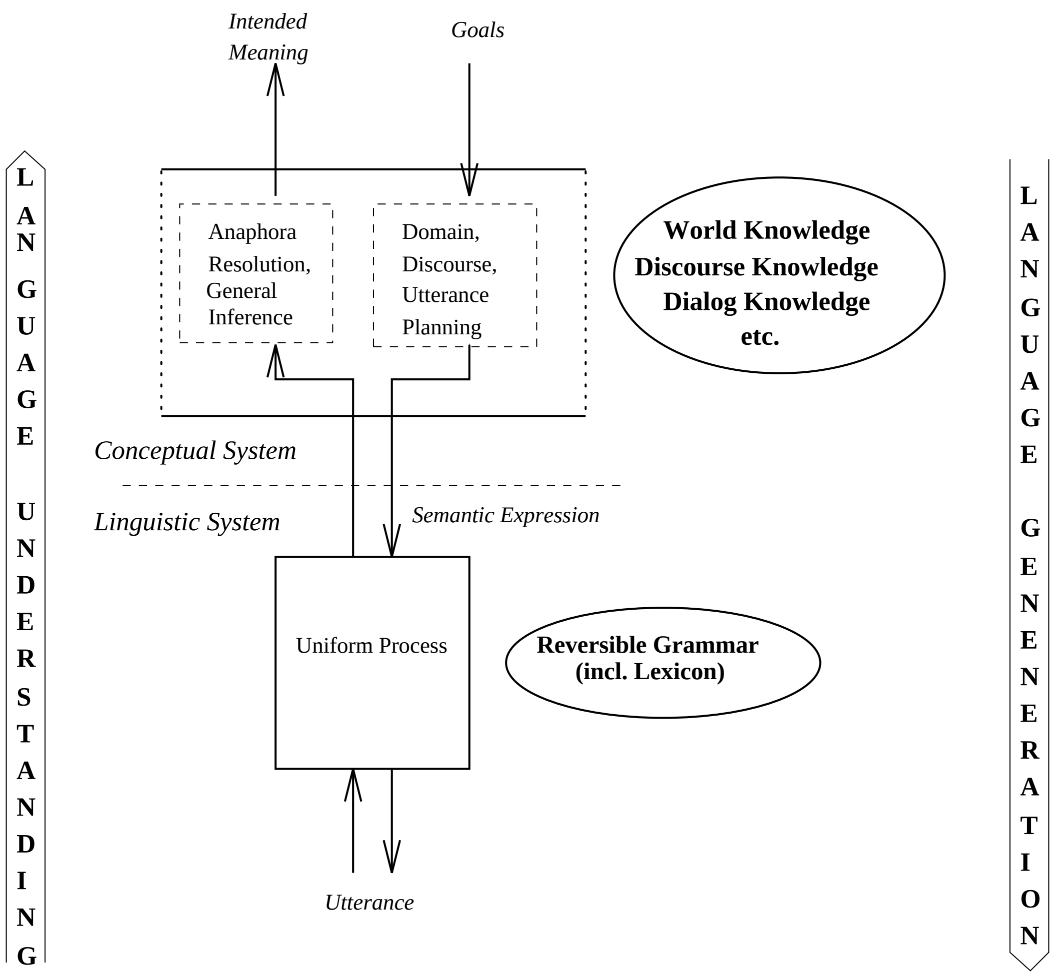


Figure 5.1: The Architecture of an NLS using a reversible grammar.

5.2 Natural Language Systems and Reversible Grammars

If we use a reversible grammar for language understanding and generation then this implies that at least with respect to the grammatical knowledge these processes must be symmetric. This implication as been pointed out by [Appelt, 1987] and we can draw the organization of a reversible system as shown in figure 5.1.

In this graphic the conceptual component is divided into two parts: one used during understanding and one used during generation. This is done in order to emphasize the different tasks to be solved during both directions. It should not exclude the possibility that during both tasks same processes, knowledge sources or formalisms can be used. Furthermore we do not want to make any claims about the whole internal structure and status of the conceptual system so we view it as an open system (indicated by the dotted lines).

In this system we assume that semantic information is represented by some kind of *logical formulas* that are used to abstract predicate–argument structure and quantifier scoping from sentences. We need not to make further assumptions about the concrete form of the logical language in order to discuss the basic claims in this section. For utterances we make the simple assumption that they are represented as strings, i.e., we represent the phonological structure of a sentence as a list of words.

The graphic in figure 5.1 makes clear that the semantic representation of the grammar serves as an intermediate representation during understanding *and* generation. For the whole system we have a clean separation between conceptual and linguistic processing in both cases (indicated by the dashed line). The linguistic system has a *modular* status because it is the only component directly concerned with grammatical knowledge *declaratively* represented in a reversible grammar. This means that while performing both tasks the same grammatical power is potentially available (regardless of the actual language use). Clearly, such a modular design has the advantages already discussed in section 2.1.

Giving the linguistic component a modular status in that way, however, implies a serious problem especially for natural language generation, namely that the conceptual component constructs a logical form only on the basis of non-grammatical knowledge while the linguistic component processes logical forms only on the basis of grammatical knowledge. This means that the conceptual component cannot control completely whether and how the linguistic system will realize a given semantic structure.

For example, the following can happen. A message which is constructed precisely enough to satisfy the conceptual component's goal can be under-specified from the linguistic component's viewpoint. In particular, the generator can *run into the risk of being misunderstood* because of the produced utterance's ambiguity. We call this the **choice problem of paraphrases**.

For example, if the conceptual component specifies the following structure SEM as input to the linguistic component:[2]

$$(12)\quad \left[\texttt{SEM}\ \left[\begin{array}{ll}\texttt{CONT} & \left[\begin{array}{ll}\texttt{RELN} & \textit{remove}' \\ \texttt{AGENT} & \textit{you}' \\ \texttt{PATIENT} & \textit{folder} \\ \texttt{INSTRUMENT} & \textit{system\_tools}'\end{array}\right] \\ \texttt{CONX} & \left[\texttt{SPEECH\_ACT}\ \textit{imperative}\right]\end{array}\right]\right]$$

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:

$$(13)\quad \left[\begin{array}{ll}\texttt{PHON} & \langle \textit{remove the folder with the system tools} \rangle \\ \texttt{SYNSEM} & S[\textit{imp}] \\ \texttt{DTRS} & \left[\begin{array}{ll}\texttt{HEAD} & \left[\begin{array}{ll}\texttt{PHON} & \langle \textit{remove} \rangle \\ \texttt{SYNSEM} & \textit{VP}[\textit{fin}]\end{array}\right] \\ \texttt{COMP} & \left[\begin{array}{ll}\texttt{PHON} & \langle \textit{the folder} \rangle \\ \texttt{SYNSEM} & \textit{NP}[\textit{acc}]\end{array}\right]\end{array}\right] \\ \texttt{ADJUNCT} & \langle \textit{with the system tools} \rangle\end{array}\right]$$

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

$$(14)\quad \left[\begin{array}{ll}\texttt{PHON} & \langle \textit{remove the folder with the system tools} \rangle \\ \texttt{SYNSEM} & S[\textit{imp}] \\ \texttt{DTRS} & \left[\begin{array}{ll}\texttt{HEAD} & \left[\begin{array}{ll}\texttt{PHON} & \textit{remove} \\ \texttt{SYNSEM} & \textit{VP}[\textit{fin}]\end{array}\right] \\ \texttt{COMP} & \left[\begin{array}{ll}\texttt{PHON} & \langle \textit{the folder with the system tools} \rangle \\ \texttt{SYNSEM} & \textit{PP} \\ \texttt{DTRS} & \left[\begin{array}{ll}\texttt{HEAD} & \langle \textit{with the system tools} \rangle \\ \texttt{COMP} & \langle \textit{the folder} \rangle\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

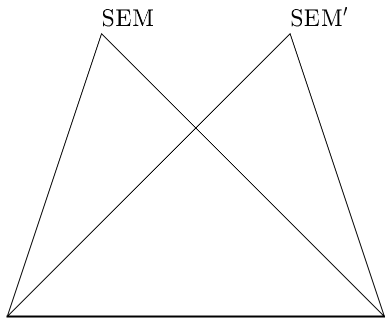
with the semantic reading SEM′:[3]

[2]We are using an HPSG-like notation close to that of [Pollard and Sag, to appear].

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

$$(15)\quad \left[\texttt{SEM}\left[\begin{array}{ll}\texttt{CONT} & \left[\begin{array}{ll}\texttt{RELN} & \mathit{remove'}\\ \texttt{AGENT} & \mathit{you'}\\ \texttt{PATIENT} & \left[\begin{array}{l}\texttt{INDET}\ x\\ \texttt{RESTR}\ \mathit{folder'}(x) \wedge \mathit{with'}(x, \mathit{system\_tools'})\end{array}\right]\end{array}\right]\\ \texttt{CONX} & \left[\texttt{SPEECH\_ACT}\ \mathit{imperative}\right]\end{array}\right]\right]$$

The whole situation can graphically be represented as follows:



Remove the folder with the system tools

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 during generation the linguistic component is mainly guided by the compositional structure of the semantic input, it cannot determine 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 conceptual 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 conceptual 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 non-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 conceptual component chooses '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 occurs: Without detailed grammatical background the conceptual component would not be able to specify the correct ordering in order to avoid ambiguity.

One might argue that when adding functional features to the feature system of a grammar (like focus, rhema, theme) in order to distinguish grammatical structures that have equal semantics but differ with respect to their functional value (cf. Fedder [1991], Bateman et al. [1992]) the problem would not occur. Consider for instance the possible realizations of topicalization in German. Topicalization can be realized using either a passive construction or by fronting movement. Assume for instance that the conceptual component wants to verbalize the following semantic feature structure:

$$(16)\quad \begin{bmatrix} \texttt{SEM} & \begin{bmatrix} \texttt{PREDICATE} & \textit{drive} \\ \texttt{AGENT} & \textit{peter} \\ \texttt{COAGENT} & \textit{maria} \end{bmatrix} \\ \texttt{PRAG} & \begin{bmatrix} \texttt{FOCUS} & + \end{bmatrix} \end{bmatrix}$$

This semantic information[4]expresses that 'Peter is the one who drives Maria'. The value of the FOCUS feature expresses that Maria is the current focus of the communication situation. Possible utterances in German are (17) and (18).

(17) Maria wird von Peter gefahren.
Maria is driven by Peter.

[4]This and the following representations of the example are simplified in order to make clear the relevant aspects of the current discussion.

(18) Maria fährt Peter.
Maria, Peter drives.

In both cases the syntactic construction can be marked by the two binary features FOCUS and EMPHASIS. In the passive case, the values of the FOCUS and EMPHASIS features would be defined as in the feature structure of (19) and for the fronting movement rule as represented in (20).

$$(19)\quad \left[\texttt{PRAG}\begin{bmatrix}\texttt{FOCUS} & + \\ \texttt{EMPHASIS} & -\end{bmatrix}\right]$$

$$(20)\quad \left[\texttt{PRAG}\begin{bmatrix}\texttt{FOCUS} & + \\ \texttt{EMPHASIS} & +\end{bmatrix}\right]$$

The problematic construction is that of (18) because during parsing there is also the unmarked reading possible, which says that **Maria** is the agent. Clearly, if the conceptual component wants to avoid misunderstandings it can choose the passive form. But to do this, it has to know that the value of EMPHASIS is necessary to distinguish both cases. But to have knowledge about this specific kind of combination of features means that it has to foresee that (18) is ambiguous. Hence it has to have detailed knowledge about the functional system and the way they are combined with specific constructions. Consequently it needs detailed knowledge about the actual grammar.

There is also psychologically grounded evidence for assuming that the input to a tactical component might not be necessary and sufficient to make linguistic decisions. This is best observed in examples of self-correction [Levelt, 1989]. For example, in the following utterance:[5]

> "but aaa, bands like aaa- aaa- aaa- errr- like groups, not bands, - groups, you know what I mean like aaa."

the speaker discovers two words (the near-synonymous 'group' and 'band') each of which comes close to the underlying concept and has problems to decide which one is the most suitable. In this case, the problem is because of a mis–match between what the conceptual component want to express and what the language is capable of expressing [Rubinoff, 1988].

It is important to note here that the problem does not arise only when using reversible grammars but is an intrinsic problem of modularity. Every natural language model that assigns the grammatical component a modular status must face the problem. An important advantage of using reversible grammar is that we can consider the problem more clearly in the case of language processing. Moreover,

[5] This example is taken from Rubinoff [1988] and is originally from a corpus of speech collected at the University of Pennsylvania.

in this thesis we demonstrate that a consistent use of reversible grammars is the starting point for solution of these problems.

Summarizing, it should be clear now that the conceptual 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 chapter 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 conceptual 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.

Currently, in generation systems where a modular design is advocated the problems are sometimes 'solved' in such a way that the conceptual component has to provide all information needed by the linguistic component to make decisions about lexical and syntactic choices [McDonald, 1983], [McKeown, 1985], [Busemann, 1990], [Horacek, 1990], [McKeown *et al.*, 1990], [Dale, 1990]. As a consequence, this implies that the input to the linguistic component is tailored to determine a good sentence, making the use of powerful grammatical processes redundant. In such approaches, linguistic components are only front–ends and the conceptual component needs detailed information about the language to use.

Hence, they are not separate modules because they both share the grammar. 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 resources (in our case the grammar).

In order to be able to handle these problems, more flexible linguistic components are necessary that are able to handle, e.g., under-specified input. In [Hovy, 1987], [Finkler and Neumann, 1989; Neumann and Finkler, 1990] and [Reithinger, 1991] approaches are described how such more flexible components can be achieved. A major point of these systems is to assume a bidirectional flow of control between the conceptual and the linguistic component.

The problem with systems where a high degree of feedback between the conceptual and the linguistic component is necessary in order to perform the generation task is that one component could not perform its specific task without the help of the other. If the linguistic component has a modular status as assumed in this thesis then it is important for a component's mode of operation that it is minimally affected by the output of another component. Levelt [1989] argues that "it makes no sense to distinguish a processing component A whose mode of operation is continuously affected by feedback from another component, B. In that case, A is not specialist anymore, it won't come up with the right result without the 'help' of B." ([Levelt, 1989], page 15). If this is the case one should better describe A and B as being one component.

5.3 Monitoring and Revision

In order to maintain a modular design additional mechanisms are necessary to perform some *monitoring* and *revision* of the generator's output. Several authors argue for such additional mechanisms [Jameson and Wahlster, 1982; DeSmedt and Kempen, 1987; Joshi, 1987; Levelt, 1989; Neumann, 1991b]. 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 thesis we will present for the first time a computational model of monitoring based on reversible grammars.

The model introduced here is based on a *strict integration of parsing and generation* in the sense of

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

Before we give a detailed algorithmic characterization of the integrated approach we will consider current approaches that also take an integrated view of natural language processing.

5.3.1 The Monitoring Model of Levelt

With very few exceptions monitoring has only given an appropriate focus of attention in cognitive science. In that area there is no denying that humans watch over what they say. In [Berg, 1986] and [Levelt, 1989] good overviews of the current state of the art are given. We will now describe the model described in [Levelt, 1989] which is currently one of the best elaborated monitoring models (although not implemented) in more detail.

Figure 5.2 shows the architecture of the monitoring model developed in [Levelt, 1989]. The language model Levelt describes follows the same modular design as discussed in section 5.1, i.e., he also divides his system into a conceptual component and a linguistic component. The linguistic component consists of two subsystems, one for production and one for understanding. The production system is further subdivided into a formulator and an articulator. The formulator receives a preverbal message from the conceptual component and produces a phonetic plan based on lexical and

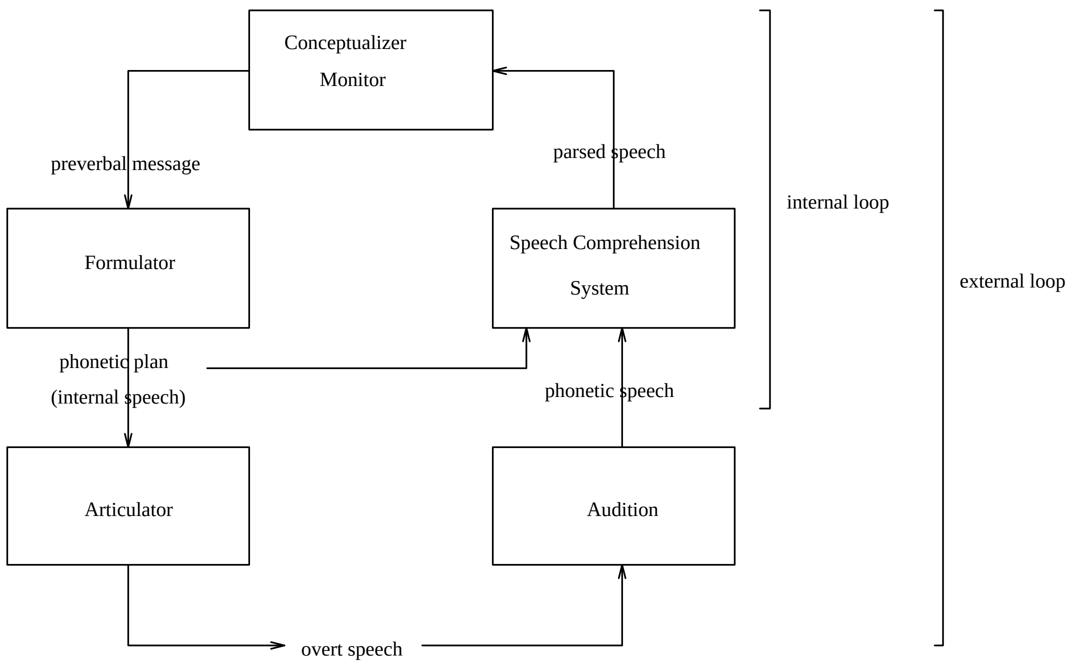


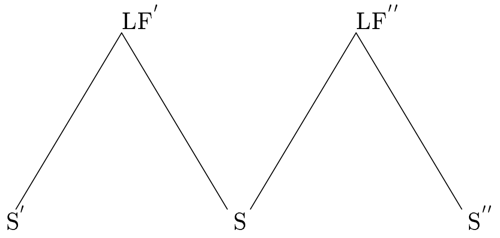
Figure 5.2: Levelt's perceptual loop theory of self–monitoring.

grammatical information. The phonetic plan is then transformed into spoken utterance by the articulator. During understanding a spoken utterance is mapped by the audition component to a phonetic string from which the speech comprehension system computes parsed speech, a representation of the input speech in terms of phonological, morphological, syntactic, and semantic composition. This representation is further processed by the conceptual component. The model introduced by Levelt is basically discussed from the production view. He therefore abstracts away from details concerning the inner working of the comprehension model.

Levelt identifies the editor with the language-understanding system to avoid reduplication. A speaker can attend to his own production in just the same way as he can attend to the speech of others (cf. [Levelt, 1989], page 469). The perceptual loop theory he discusses consists of a double "perceptual loop" to model that a speaker

- can attend to his own *internal* speech before it is uttered and
- that she can also attend to her self–produced *overt* speech.

Discussion In Levelt's model parsing and generation are performed in isolation using two different grammars (although he only considers generation in full detail). 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. If, for example an intelligent help-system that supports a user by using an operating system (e.g. Unix, [Wilensky *et al.*, 1984]), 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 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 summarized in the following figure (LF' and LF'' symbolize 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 5.3: Relationship between ambiguities and paraphrases.

If parsing and generation are performed in isolation then generation of paraphrases can be very inefficient, because the source of the ambiguous utterance S is not used directly to guide the generation process.

5.3.2 The Anticipation Feedback Loop Mode

In the area of natural language systems a similar method that also integrates understanding and generation like Levelt’s model is known under the term of *anticipation feedback loop* (AFL) which has been motivated as a special case of exploitation of a user model [Wahlster and Kobsa, 1986]. The basic idea of the AFL model is the use of the system’s natural language understanding part to anticipate the prefered user’s interpretation of an utterance which the system plans to realize. User-modelling is necessary to answer a question like (cf. [Wahlster, 1991])

> If I had to analyze this communication act relative to the assumed knowledge of the user, then what would be the effect on me?

To answer such question a produced utterance is fed back to the system's NLU part under the consideration of the user model. If the result of the understanding process does not match the system's intention in planning, it has to re-plan its utterance. Figure 5.4 shows the schematic structure of a system which incorporates an anticipation feedback loop.

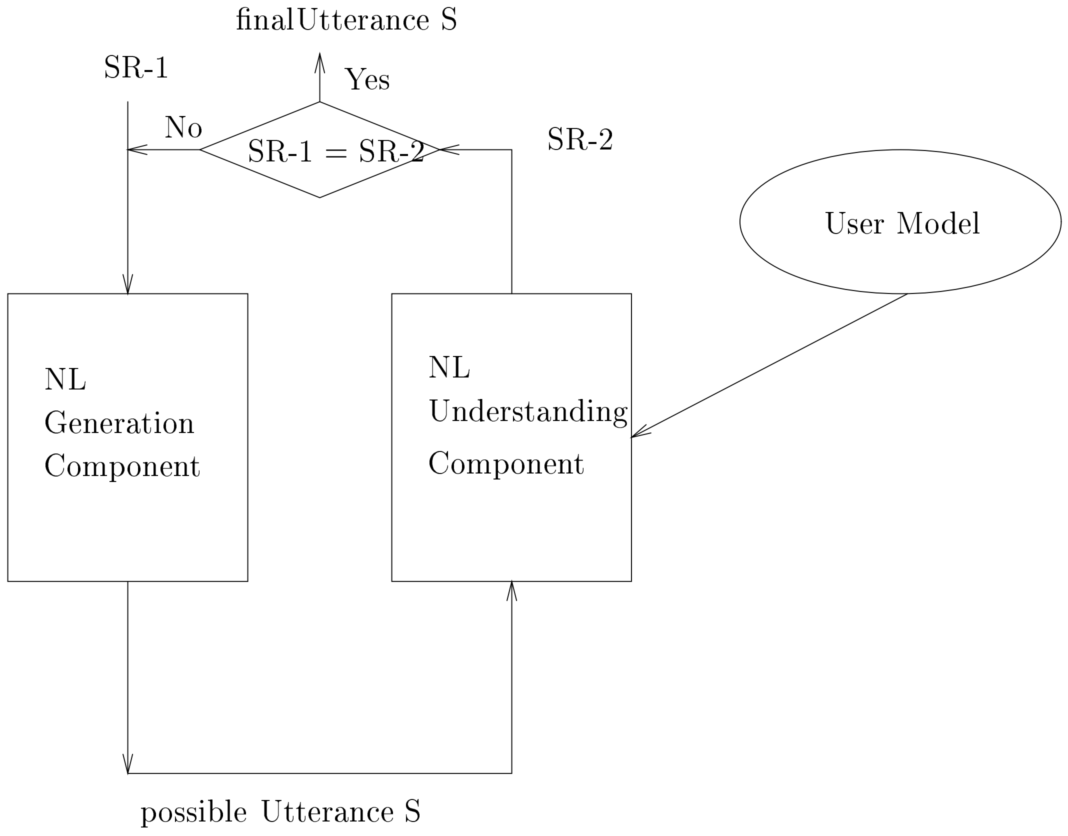


Figure 5.4: Schematic structure of an Anticipation Feedback Loop, based on Wahlster and Kobsa [1986].

A possible utterance S generated from a semantic representation SR_1 is fed back to the understanding component of the system. The result SR_2 has been computed under consideration of the user model. If this result does not match the original goal SR_1 the system interprets this as a possible source of misunderstandings. Therefore SR_1 is iteratively revised until the analysis of the produced utterances matches the system's intention.

In [Jameson and Wahlster, 1982] this method was used for generating elliptical utterances in the HAM–ANS system. A local anticipation feedback loop is used to ensure that the system's generated ellipses are not so brief as to be ambiguous or misleading. Suppose for example the user as entered the question:

(21) Are there three beds and at least one desk in the room?

If the room has in fact three beds and three desks, an appropriate answer of the system would be

(22) Yes, three beds and three desks

instead of one of the possible other answers like

(23) Yes.
Yes, three.

because they are undetermined with respect to correct interpretation. Therefore, the semantic representation SR_Q of the user's question is taken into account during the process of ellipsis generation in the following way. After constructing the complete semantic representation SR_A of the system's answer, SR_A is compared from the top downward with SR_Q in order to determine the set of essentially identical subtrees of both representations. The top-down approach orders them automatically with respect to their size. The smallest identical subtree is chosen as a possible candidate for an elliptic utterance. Before that partial semantic representation SR_{part} is actually passed to the surface transformation process, SR_{part} is fed back to the system to check whether SR_{part} will possibly be understood by the user according to the system's intention. Using the ellipsis reconstruction component of the system's understanding part, SR_{part} is compared with SR_Q to be able to determine whether SR_{part} is actually a part of SR_Q (which must be the case because otherwise the system itself is in an inconsistent state). If SR_{part} occurs only in one subtree of SR_Q then it is chosen as a candidate and passed to the verbalization component. If two or more subtrees exist in SR_Q that match SR_{part}, SR_{part} is rejected and the next larger subtree of SR_A is chosen from the list of possible candidates and the same method is applied again. In summary, before verbalizing an elliptic utterance immediately, the system attempts to reconstruct its semantic representation as the user would, i.e. by determining how it fits into the structure of the original question SR_Q.

Discussion In the method described above the anticipation feedback loop operates only on the semantic level, i.e. whether an elliptic utterance will be ambiguous or not is determined only under semantic considerations. If a candidate partial subtree has been found the grammatical construction of it is done without checking the utterance's grammatical ambiguity. The implicit assumption made here is that the string determined for an unambiguous SR_{part} will also be analyzed definitely as SR_{part}. But this does not avoid the risk of being misunderstood in general, because it could be the case that the resulting string is still ambiguous. Most interesting from a reversibility standpoint is that Jameson and Wahlster [1982] also vote for the use of reversible knowledge sources

> . . ., the recognition component must operate on the same kind of structure as those returned by the system's ellipsis generation component.

In HAM–ANS the generation and understanding components share the same internal representation language. However, two different grammars are used during parsing and generation. If AFL were also designed for grammatical processing (actually they do not consider integration of parsing and generation) the same problems as already mentioned for Levelt's model would also occur.

5.3.3 Text Revision

In [Vaughan and McDonald, 1986] and [Meteer, 1990] a model of text revision in natural language generation is proposed that is based on an integrated approach to parsing and generation. It is based on the observation that during composition of a text, a multi-pass system of writing and rewriting is used to produce an optimal text. They describe a model that includes a generator, a parser and an evaluation component which assesses the parse of what the generator had produced and determines strategies for improvement. The revision process they outline is modelled in terms of the following three phases:

1. recognition, which determines where there are potential problems in the text;
2. editing, which determines what strategies for revision are appropriate and chooses which, if any, to employ;
3. re–generation, which employs the chosen strategy by directing the decision making in the generation of the text at appropriate moments.

The recognition phase is responsible for parsing text and building a representation rich enough to be evaluated in terms of how well the text coheres. The text representation they developed is called *text structure* and serves as intermediate level of representation of text planning (cf. [Meteer, 1990]). The recognition phase analyzes the text as it proceeds using a set of evaluation criteria (e.g., to find ambiguous referents, flag places where optimizations may be possible, such as predicate nominal). For each problem there is a set of one or more strategies for correcting it. The task of the editing phase is to determine which of these strategies to employ (e.g., if the subject has a relation to a previous referent which is not explicitly mentioned in the text, more information may be added through modification). The final step is actually making the change once the strategy has been chosen. This essentially involves marking the input to the generator, so that it will query the revision component at appropriate decision points. For example, if the goal is to put two sentences into parallel structure, the generator when reaching this marker asks the revision component whether it should realize two main clauses or if it should realize one as a subordinate and how it should be realized (e.g., active or passive).

Discussion It is very difficult to compare the model in detail with the model described in this thesis because "A similarity between Penman's revision module and the model described in this paper is that neither has been implemented." ([Vaughan and McDonald, 1986], page 95). Although Meteer [1990] gives a detail description of the relationship between *text structure* and revision it is unclear how the proposed model could contribute to the choice problem of paraphrases (see section 5.2). However, from the approach described above and from the system described in [Meteer, 1990] we can draw the following conclusions. Only the generator's input is marked. If the generator encounters alternative realizations the revision component is asked to make the decision. However, to be able to do this it needs detailed knowledge about the grammar. Therefore grammatical knowledge has to be duplicated. The linguistic realization component used in [Meteer, 1990] is MUMBLE-86 [McDonald, 1986]. The text structural representation level must completely specify the information to be expressed by the utterance. The mapping has to ensure that all the necessary linguistic information is present. Mumble's *procedural* grammar is used only for generation purposes. Therefore it is without reach for the revision model to take into account relevant sources of ambiguities.

5.4 A Blueprint of the New Model

The model that we use for this approach basically consists of three components:

- the kernel linguistic component , subdivided into a reversible grammar plus lexicon and a parser/generator
- the common knowledge pool for derivation trees
- the editor that performs comparison and revision

The reversible system introduced in section 5.2 (see figure 5.1) can be represented more detailed as shown in figure 5.5.

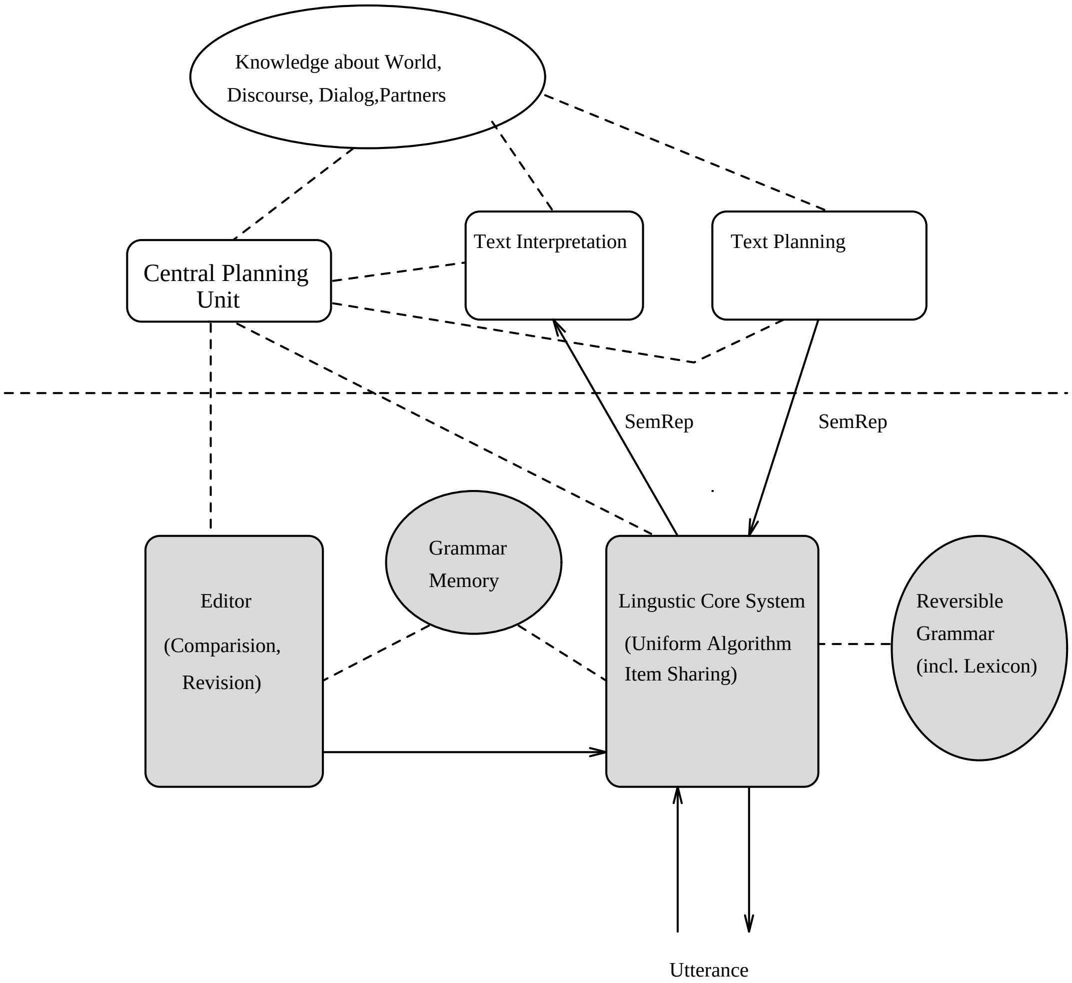


Figure 5.5: The Architecture of the new reversible system.

The linguistic core system consists of a reversible grammar (plus lexicon) and the uniform algorithm used for parsing or generation. Both together constitute the *competence base* of a natural language system. Thus, we are following an extended competence description as known from cognitive linguistics (see e.g., [Schwarz, 1992]), such that we do not only consider the grammatical knowledge as the only source of language competence (cf. [Chomsky, 1986]), but also those processes that are able to determine all possible well–formed grammatical structures of a language, at least potentially. However, during run-time of a system we assume that only a subset of possible solutions are computed. This makes sense at least in the case of generation because in practice only one utterance should be produced for a given semantic representation. In our view, the methods or techniques that are used to determine most adequate readings or paraphrases in some specific situation, or those that are used to perform for instance some kind of robustness or monitoring, belong to the performance issue of language use.

The derivations obtained during parsing and generation are stored in a common memory which we will call *common knowledge pool* for derivation trees, abbreviated as CKP.

The editor is that part of the system that compares structures obtained during monitoring and stored in CKP and eventually revises previously computed solutions. We assume that the editor has a functional nature consisting of a set of specific functions for performing particular tasks (e.g., generation of unambiguous utterances or paraphrases). These functions are selected according to the major flow of control (either understanding or generation) and in dependence of the specified goals, e.g., 'be as unambiguous as possible', 're-solve ambiguities explicitly'.

Monitoring is not performed by a separate module. Instead the understander or generator fulfill this part (either during generation or understanding). The basic task of monitoring is to gain information about processing which is not necessarily obvious, i.e., a device is called for which this information can be available to the speaker or the hearer. In has often been argued in cognitive psychology (cf. [Berg, 1986; Levelt, 1989]) that it is highly desirable to find a mechanism that is an integral and independently motivated part of the whole system and one that performs the monitoring function by its own nature. Clearly, if it is possible to use devices that are needed anyway for natural language processing, this fulfills our effort to avoid redundancies and to obtained a certain degree of economy.

Whether monitoring and editing should take place is decided by a central planning unit which is part of the conceptual component. We assume that the degree of monitoring and editing that takes place during processing depends on the degree of attention a system has in ongoing communications. This depends on the relevance of the dialog contribution and on the specific dialog situation. For example, if monitoring is switched on by the central planner then it depends on the relevance of the information to be uttered if editing should take place. If the system is under time pressure then the planner might decide to avoid editing because it would take to much time in order to utter the information immediately. Or if the editor cannot find an un-ambiguous reading then the planner has to decide whether to recompute the communication goal that possibly can be mapped to an un-ambiguous utterance (actually the planner would delegate the task of re-computation of the original plan to the text planner; this is indicated by the dashed line in figure 5.5.). The central planning unit will not be further investigated in this thesis because we are mainly concerned with the realization aspects of monitoring and editing on the grammatical level. However, as mentioned above we assume the existing of this unit as the primarily activator for monitoring.

The integrated approach developed in this thesis, is basically used for revision of previously produced utterances. These utterances are assumed to be grammatically well-formed. The integrated approach is used in order to reduce the risk of being misunderstood in the case of the utterance's ambiguity. However, monitoring can also be very useful for detection and correction of speech errors. There is psychological evidence that monitoring mechanisms are also involved in the case of self-repair of lexical errors (using wrong lexemes for a known concept), errors forced by trouble during phonological encoding (wrong spelling or prosody), slip of the tongue phenomena or errors on morphological and grammatical level [Levelt, 1989]. In this thesis we do not consider self-repair of utterances containing errors. In principle, it could be questionable, whether a computational model of such error handling is of interest not only for cognitive psychology but also for the development of NLS, in general. However, the detailed discussion of this question is beyond the scope of this work.

What does this model contribute to system design? To use Gerhard Kempen's words (cf. [Kempen, 1989], page 15)

> Nevertheless, the addition of a monitor may contribute to the solution of practical and theoretical problems significantly. Take for example the above issue of one-way versus two-way traffic between strategic and tactical components.[6] Suppose the monitor can intercept the linguistic output from the tactical component (preferably before the point of speech) and feed it into a parser/understander. The latter evaluates the generator's utterance from relevant viewpoints and informs (via the monitor) the strategic component of its diagnosis. This would establish the line of communication postulated by Danlos and others without complicating the generator's design — the parser is needed anyway.

In a similar vein, Mann [1987], page 207 suggests that

[6] In the authors term, the strategic component corresponds to the conceptual system and the tactical to the linguistic system.

> More substantial sharing occurs in the area of knowledge representation and inference. Here the problems and solution, not just the recognition of phenomena, are shared. There is hope for convergence, for one all-sufficient underlying representational form, and for a non-directional view of language. It is often suggested that an adequate text generator must have an understander inside to check its work. Still, the research activity is dominated by the differences rather than the shared elements.

Putting both citations together our approach can be seen as an important step in that direction. We have to incorporate into language interfaces the same kind of sensitivity to later audience reactions that we have ourselves. If we are able to do this, then this will lead to more flexible natural language systems. Clearly, in order to get a real-time behaviour of our systems, the amount of feedback between the conceptual and linguistic system should be reduced as far as possible. If the evaluation of some decision points cannot be performed deterministically with information of the input the system should be free to choose according some preferences (for example 'the first alternative you can get' or 'take that with the highest priority'). However, if we are in a particular situation, then the conceptual component especially the monitor should be able to delegate the appropriate goals to obtain a more carefully realization to the linguistic component. This component is now responsible for doing its best. By means of monitoring the conceptual component can be part of this process in that it can observe the input/output behaviour of the linguistic system and evaluates the results with respect to the specified goals.

This kind of modelling means for the design of a whole system, that the particular goals that have to be fulfilled in order to obtain adequate realization can be formulated in a discourse independent abstract way. Because during run-time the conceptual and linguistic component are able to solve the goals in a cooperative way, the overall system is able to react on particular situations more flexible during run-time. If we would deny the importance of such mechanisms for the investigation of NLS then we have to foresee in a system the creative aspect of possible situations and hence, we are forced to specify the flexibility by hand.

5.5 Monitoring and Revision with Reversible Grammars

In this section we present a method for self-monitoring and revision with reversible grammars that does not depend on the use of any specific generation and parsing algorithm. We therefore use *generate* and *parse* as abstract names for the algorithms used. In section 5.5.8 we describe how specific methods (including the uniform algorithm) can be adapted for use in the monitoring strategy.

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

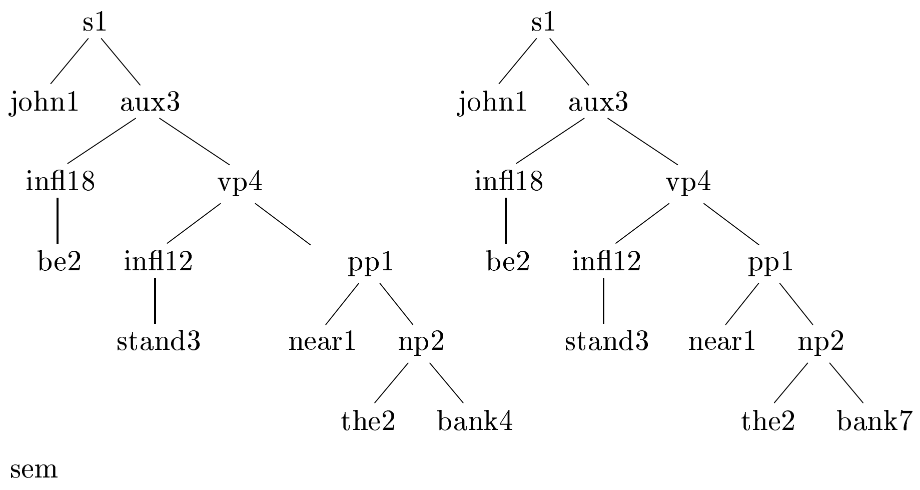


Figure 5.6: Derivation trees

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' and 'money institution') a generator may produce, as a first possibility, the following sentence in the case of the first reading of 'bank'.

(24) John was standing near a 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:

(25) 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 Croatian army]' 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.

5.5.1 Locating Ambiguity with Derivation Trees

We hypothesize that a good way to characterize 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.

We assume that a derivation tree is represented as part of the feature structure of a linguistic sign as introduced in chapter 3, section 3.3. However for convenience, we will represent derivation trees using the well-known tree-like representation.

For example, the derivation trees for the two readings of 'John is standing near the bank' may look as in figure 5.6. 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`.

5.5.2 Overview of the Monitored Generation Strategy

The derivation trees obtained during generation and parsing will be used for *guiding* the monitored generation strategy in the following way: Given a logical form the normal generator first produces only one string (instead of all possible strings). Only if the overall system specified as a primarily goal that the produced string should be unambiguous (in order to avoid the risk of being misunderstood), the monitored generation strategy is activated. It first passes the produced string (say α) to the parser. If the parser yields severals readings, the obtained parsed derivation trees are now used for comparison with the generated derivation tree of α. This derivation tree is marked at that places, where in corresponding places of the parsed derivation tree, different rules have been applied. It is assumed that this indicates a source of structural ambiguity. Therefore, the semantics of the root node of such a marked subtree is revised using a different rule from that of the original generated derivation tree. This will eventually create a new string α'. This 'new' string is then also revised, if it is still ambiguous.

If we want to make use of the representation of derivation trees as introduced in chapter 3, section 3.3, we have to make sure that we can retrieve for each node of a derivation tree its local semantics and string. However, this is easily obtained by adding two new features SEM and PHON to the derivation tree of grammar rules and lexical entries whose values are just pointers to the values of the corresponding features of the sign. For example, rule (r_1) of the grammar in appendix A is modified as follows:

$$
(r_1)\quad sign\left(\begin{bmatrix}
\texttt{cat:} & vp \\
\texttt{sc:} & Tail \\
\texttt{sem:} & Sem \\
\texttt{lex:} & no \\
\texttt{v2:} & V \\
\texttt{phon:} & {}_{Str}\ P_{0}\text{-}P \\
\textsc{deriv} & \begin{bmatrix}
\texttt{rulename} & vp\text{-}sc \\
\texttt{sem:} & Sem \\
\texttt{phon:} & Str \\
\textsc{dtrs} & \langle D_1, D_2\rangle
\end{bmatrix}
\end{bmatrix}\right) \longleftarrow
$$

$$
sign\left({}_{Arg}\begin{bmatrix}
\texttt{v2:} & no \\
\texttt{phon:} & P_{0}\text{-}P_{1} \\
\textsc{deriv} & D_{1}
\end{bmatrix}\right), sign\left(\begin{bmatrix}
\texttt{cat:} & vp \\
\texttt{sc:} & \langle Arg|Tail\rangle \\
\texttt{sem:} & Sem \\
\texttt{v2:} & V \\
\texttt{phon:} & P_{1}\text{-}P \\
\textsc{deriv} & D_{2}
\end{bmatrix}\right)
$$

The advantage of this kind of representation is that for each node of a derivation tree we know which part of the input (either semantics or string) is covered by the corresponding sub-derivation. Now, if a node has been marked as an ambiguous source, it is easy to call the normal generator for the marked node's semantic expression in order to produce an alternative string. The next paragraph describes how a marked derivation tree is obtained.

5.5.3 Marking a Derivation Tree

Given a derivation tree t of a generated sentence s, we can 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 (25) 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 5.7.

We will define a procedure MARK that marks the generated tree given the trees found by the parser. Marking a node will be done by adding the feature MARKED with value YES in addition to features LABEL and DTRS. If a node has been marked

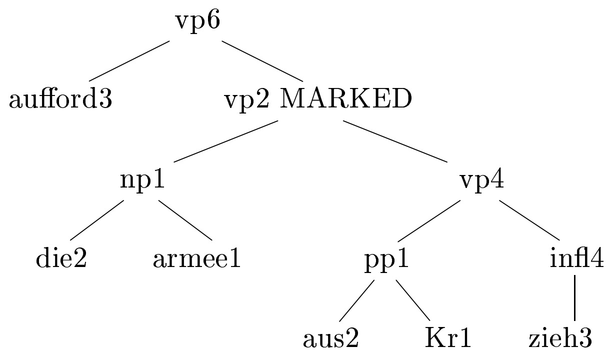


Figure 5.7: Marked derivation tree

then scanning of its subtree will stop, i.e., the nodes of a subtree with a marked root will not be marked. Thus, the definition of the procedure MARK is as follows:

```
mark(Tree, TreeSet):
 if root_same(label(Tree),TreeSet)
   then mark_ds(dtrs(Tree),get_ds(TreeSet))
   else
    return mark_node(Tree).

root_same(Label,TreeSet):
 if empty(TreeSet)
   then return true
  else
   if equal(Label,label(first(TreeSet)))
    then root_same(Label,rest(TreeSet))
     else return false.
```

We first compare the rule names of each root node of trees in the set of parsed trees TreeSet with the rule name of the root node of the generated tree Tree, using the function ROOT_SAME. If all are the same (i.e., the same rule has been applied during generation and parsing), we conclude that no ambiguity has occurred at that level so we scan the next level of the trees in parallel, using the function MARK_DS, where the function call DTRS(TREE) just returns the daughter trees of Tree, and the call GET_DS(TREESET) returns a list containing all daughter trees of all trees in

TreeSet preserving the relative order. More precisely, we return a list of lists where the i-th list contains the i-th daughter of each tree in TreeSet. Within the function MARK_DS the function MARK is called recursively on each daughter tree of Tree.

If, however, the root nodes are not equal, we return the Tree after having marked the root node of Tree using the function MARK_NODE. This function destructively adds the feature MARKED with value YES.

5.5.4 Changing the Ambiguous Parts

Generation of an utterance given a marked derivation tree informally 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 function *mgen* is used to generate an utterance, using a marked derivation tree as an extra guide.

```
mgen(MarkedTree):
 if marked_node(root_node(MarkedTree))
  then generate(semantics(root_node(MarkedTree)))
  else apply_rule(root_node(MarkedTree));
       mgen_dtrs(dtrs(MarkedTree)).
```

```
mgen_dtrs(Dtrs):
 if empty(Dtrs)
  then return T
  else
   mgen(first(Dtrs));
   mgen_dtrs(rest(Dtrs)).
```

This function scans a marked tree, and if it encounters a marked node it just calls the normal generator GENERATOR with the semantic expression of the marked node[7]. For each unmarked we redo the previous made computation of the normal generator and proceed by scanning the subtrees of the next level.

[7]In section 5.5.8 we give some more details how the actual used normal generator might be embed into the monitoring strategy.

5.5.5 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 strategy 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 strategy 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 grammatically 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. We will assume the definition of a function named MARK_L_G for pushing markers one level up. Now, the whole algorithm can be completed as follows.

```
monitored_generation(goal):
  GenTree := generate_one(goal),
  TreeSet := find_all_parses(string(root_node(GenTree)))
  if null(rest(TreeSet))
   then string(root_node(GenTree))
   else Guide := mark(GenTree,TreeSet);
        revision(Guide)

generate_one(goal):
 ``just compute the first string and then stop''

find_all_parses(string):
 ``find all readings for a given string;
   ignore spurious ambiguities, in such a way
   that if two parsed trees have the same semantics
   then retain only one.''

revision(Guide):
 NewRes := mgen(Guide)
 if unambiguous(NewRes)
  then string(NewRes)
  else revision(mark_l_g(Guide)).
```

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, eventually pushing the markers successively upwards.

5.5.6 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 (25).

(26) 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 5.8.

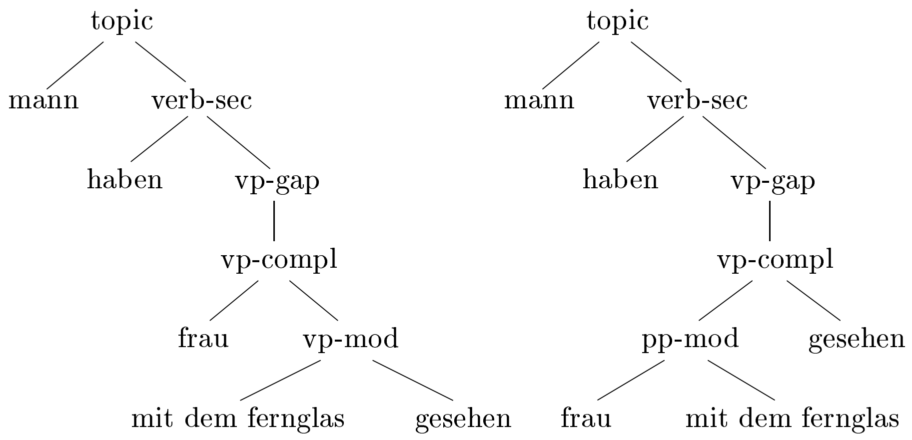


Figure 5.8: Derivation trees of the simple attachment example

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 5.8 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 5.9.

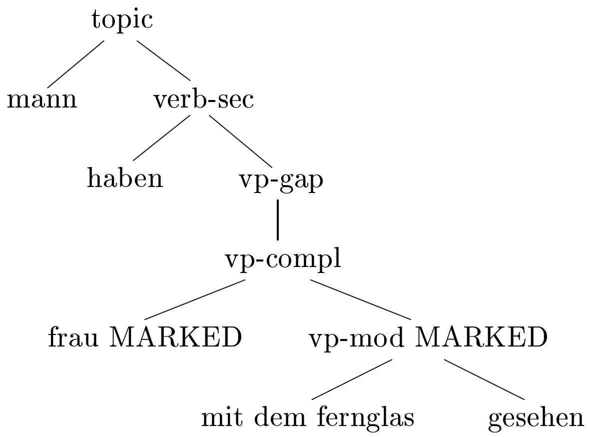


Figure 5.9: Marked tree of German example

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 5.10.

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

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

At this point we can 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

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

5.5.7 Some More Examples

In the same principle way the monitoring algorithm is able to generate the following pairs of ambiguous and unambiguous sentences. These examples have been produced by a Prolog version of the monitored generation strategy used in the PlayMoBiLD system – developed as part of the project BiLD (short for Bidirectional Linguistic

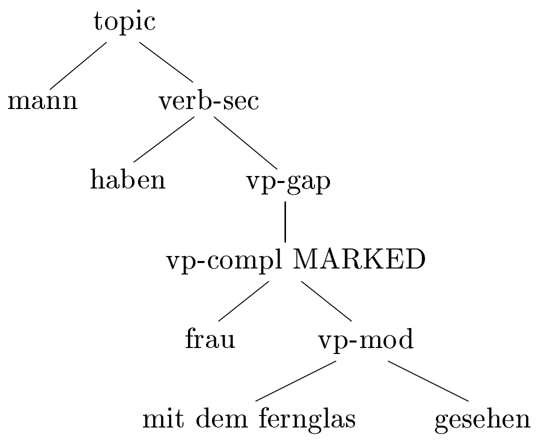


Figure 5.10: Markers are pushed one level upward

Deduction) [Neumann, 1991a].[8] The embedded generator and parser are described in [VanNoord, 1993]. The systems runs with a lexicalized grammar for Dutch (written by Gertjan van Noord, which is in its style similar to the one given in appendix A, but covers more constructions) and with a lexicalized grammar for German written by Wojciech Skut.

For both grammars, it is the case that the primarily motivation of their design and form of representation is a linguistic one, i.e., they have not been realized "with monitoring in mind". This is important, because otherwise one could criticize that the grammar "meets" the particular problem specification, and therefore might only be use-able for that specific purpose.

We will give first some examples produced by the monitored generator using the Dutch grammar:[9]

[8] The BiLD project is supported by the German Science Foundation in its Special Collaborative Research Program on Artificial Intelligence and Knowledge Based Systems SFB 314. The project location is the department of Computational Linguistics at the University of Saarland, and the principle investigator is Hans Uszkoreit.

[9] We also give English literal and (hopefully) correct translations. It should be clear that the English translations need not necessarily be ambiguous or unambiguous in the same way.

input semantics:	[schijnt([ziet([bob],[arie])])]
ambiguous string:	bob schijnt arie te zien
	**bob seems arie to see*
	bob seems to see arie
unambiguous string:	het schijnt dat bob arie ziet
	**It seems that bob arie sees*
	It seems that bob sees arie

input semantics:	[schijnt([perf,vertelt([arie],[slaapt([bob])],[jan])])]
ambiguous string:	jan schijnt arie te hebben verteld dat bob slaapt
	**jan seems arie to have told that bob sleeps*
	jan seems to have told arie that bob sleeps
unambiguous string:	dat bob slaapt schijnt arie jan te hebben verteld
	**that bob sleeps seems arie jan to have told*
	that bob sleeps seems arie to have told jan

input semantics:	[[met([bob])],schijnt([ziet([jan],[man,plur])])]
ambiguous string:	met bob schijnt jan de mannen te zien
	**with bob seems jan the men to see*
	jan seems to see bob with the men
unambiguous string:	met bob schijnt het dat jan de mannen ziet
	**with bob seems it that jan the men sees*
	it seems that jan sees the men with bob

Next some German examples:

input semantics:	past(bringen(the(mann(singular(Z6))), peter(singular(P8)), the(geldinstitut(singular(L8)))))
ambiguous string:	der mann hat peter zur bank gebracht
	**the man (NOM) has peter to-the bank brought*
	The man has brought peter to the bank
unambiguous string:	der mann hat peter zum geldinstitut gebracht
	**the man has peter to-the money-institution brought*
	The man has brought peter to the money institution

input semantics:	lieben(hans(singular(V5)),maria(singular(G7)))
ambiguous string:	hans liebt maria
	*_hans (NOM) loves mary_
	peter loves mary
unambiguous string:	maria liebt der hans
	*_mary loves the hans (NOM)_
	mary, hans loves

input semantics:	sehen(exists(spion(singular(L6))), the(mann(singular(D8))), the(fernrohr(singular(Z7))))
ambiguous string:	ein spion sieht den mann mit dem fernrohr
	a spy sees the man with the telescope
unambiguous string:	der mann wird mit dem fernrohr gesehen von einem spion
	*_the man is with the telescope seen by a spy_
	the man is seen by a spy with the telescope

Although, these successfully processed examples must be considered relative to the coverage of the grammars in use, they give a sense of the potential power the monitored generation strategy bears.

5.5.8 Properties and Implementation

Properties Some of the important properties of the non-incremental 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 of 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.

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.

Implementation A Prolog version of the monitored generation strategy is described in [Neumann and van Noord, 1992]. In this version, a semantic-head-driven generation algorithm is used for normal generation, and for parsing a head-corner parser is used. A detailed description of both algorithms can be found in [VanNoord, 1993].

Using the bottom-up generation algorithm SHDGA, however, has the disadvantage that when SHDGA is called for revision this can only be performed in a kind of generate-and-test mode. The problem is that SHDGA basically constructs a derivation tree in a bottom-up fashion. On the other side, revision means to choose an alternative rule for a marked node, which is the root node of the subtree spanned by the ambiguous substring. But then, SHDGA must first produce a candidate alternative string and an alternative subtree respectively, before it can check, whether the root node of the new subtree is in fact an alternative. This means, that after calling SHDGA an additional test is necessary which compares the rule name of the marked node with that of the newly generated subtree. The following is the relevant snapshot of the Prolog code presented in [Neumann and van Noord, 1992]:

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

Here, the term t(Name,Ds,y) represents the (local) derivation tree of a sign, where the symbol y indicates that the root node of this derivation tree has been marked for revision. First, the embedded generator is called, and only if the resulting (local) derivation is not the same the resulting string of revision is accepted. Otherwise (by means of backtracking) another possibility is tried.

Using our uniform algorithm this computational overhead is easily avoided, because of its top/down and goal-directed behaviour. For the uniform algorithm to be usable for the monitored generation strategy we only need to pass the rule name of a marked node as an additional argument to the generation mode. The predictor rule then uses this rule name as a requirement to ignore it. Furthermore, the selection strategy used by the agenda should be depth-first or best-first, and when a string has been found, the generator is forced to stop further computation. Instead of giving more details, we will directly embed the uniform algorithm in the incremental monitored strategy. Before that, however, we will show in the next section how the above described generation method can be used for the generation of paraphrases during the understanding mode of a natural language system.

5.6 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 5.3 fig. 5.3. 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 systems tools" or "Remove the folder that contains the system tools"?').

5.6.1 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. 5.3. 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 3.4) 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 5.2). 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.

5.6.2 A More Suitable Strategy

A more suitable strategy would be to generate only one paraphrase for each ambiguous logical form. As long as parsing and generation are performed in an isolated way the problem with this strategy is that there is no control over the choice of paraphrases. In order to make clear this point I will look closer to the underlying structure of the example's utterances.

The problem why there are two readings is that the PP 'with the system folder' can be attached into modifier position of the NP 'the folder' (expressing the semantic relation that 'folder' contains 'system tools') or of the verb 'remove' (expressing semantically that 'system tools' is the instrument of the described situation). The example feature structures 13 and 14 (see section 5.2) show the internal grammatical structure in a HPSG–style notation (omitting details that are not relevant in this context).

As long as the source of the ambiguity is not known it is possible to generate in both cases the utterance 'Remove the folder with the system–tools' as a paraphrase of itself. Of course, it is possible to compare the resulting strings with the input string S. But because the source of the ambiguity is not known the loop between the isolated processes must be performed several times in general.

A better strategy would be to recognize relevant sources of ambiguities during parsing and to use this information to guide the generation process. Meteer and Shaked [1988] 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 to 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 need to be paraphrased S has to be parsed again. During this repeated parse *all* potential ambiguities have to be recognized and recorded (i.e. have to be *monitored*) 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.

5.6.3 A Suitable Strategy

The crucial point during the process of generation of paraphrases is that one not only has 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.

Because most of the functions to use are already defined in section 5.5 we can directly specify the top-level function `interactive_parsing` as follows:

```
interactive_parsing(goal):
 TreeSet := find_all_parses(string(goal))
 if card(TreeSet) = 1
  then return semantics(root_node(first(TreeSet)))
  else
   Paraphrases := generate_paraphrases(TreeSet);
   ask_best_answer(TreeSet,Paraphrases).

generate_paraphrases(TreeSet):
 for each Tree in TreeSet do
  collect
   generate_paraphrase(Tree, TreeSet/{Tree})
   in Paraphrases
  finally return Paraphrases.

generate_paraphrase(Tree,TreeSet):
 Guide := mark(Tree,TreeSet);
 revision(Guide).
```

The predicate FIND_ALL_PARSE computes all possible parses TREESET of a given string (note, that this function also deletes spurious ambiguities). If the parser obtains multiple interpretations (i.e., the cardinality of TreeSet is greater than one) then for each element of TreeSet a paraphrase has to be generated. This is done by means of the predicate GENERATE_PARAPHRASES, whose explanation 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 reading determines the result of the paraphrasing process.

For each parsed sign of the form a paraphrase is generated in the following way: First its derivation tree TREE is marked by means of the set of derivations trees contained in `TreeSet/Tree`. 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.

5.6.4 A Simple Example

In order to clarify how the strategy works we consider the attachment example of section 5.5 again. Suppose that for the sentence

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

the parser has determined the derivation trees in figure 5.8 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

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

is generated in the same way described in section 5.5. In this case the left tree of figure 5.8 is marked by means of the right one.

In order to yield a paraphrase for the second reading, the right derivation tree of figure 5.8 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

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

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

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

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.

5.6.5 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:

(33) Den Studenten hat der Professor benotet, der das Programm entwickelte.
The-ACC student-ACC has the professor-NOM graded, 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:

(34) Der Professor hat den Studenten benotet, der das Programm entwickelte.
The professor-NOM has the-ACC student-ACC graded, 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:

(35) Den Studenten, der das Programm entwickelte hat der Professor 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 [1988], but rather use the parser to detect possible ambiguities. Hence our approach is much more independent of the underlying grammar.

5.7 Incremental Interleaving of Parsing and Generation

A fundamental assumption of the non-incremental version described so far is that it is often possible to change an ambiguous utterance *locally* to obtain an unambiguous utterance with the same meaning. Based on this local view it seems plausible to integrate parsing and generation more tightly in the following way: During generation already produced partial strings are parsed to determine the degree of ambiguity. If necessary an ambiguous partial string is revised in order to produce an unambiguous paraphrase of that ambiguous partial string. The successive application of this *incremental generate, parse and revise* technique will end up in an utterance which is unambiguously as possible.

Such a strategy works for an example like:

(36) 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

(37) 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

(38) 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 phenomenon 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

(39) 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

(40) I know of no better ball.

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

5.7.1 Basic Problems of Incremental Monitoring

The problem is that the monitor must be dynamically configured during incremental processing time of single utterances in order to decide

- when the test of ambiguity should take place and
- which partial strings should be revised?

Technically it is possible to check and revise each partial result 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.

Clearly, 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.

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 [Shieber *et al.*, 1990].

For example, assume that for each recursive call to the generator the revised monitor is called with an extra argument `Head` which is to be used as contextual information when the embedded parser is called to test whether the string in question is ambiguous. 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 an 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.

Beside the fact, that this method depends on grammar theories which comes with a notion of head, the main problem of this approach is that

- either each ambiguous partial string has to be revised immediately or
- revision is delayed until the previous recursive call of the generator has been finished.

In the first case the monitor would also revise irrelevant ambiguities. The latter point would mean that revision can only be performed after the whole utterance has been produced. But then the incremental method just simulates the non-incremental method.

5.7.2 A Look-Back Strategy

It seems to be more plausible to test the ambiguity of a partial string with respect to already produced partial strings. Based on this idea the notation of context is considered as follows: The *context* of a partial string α with constituent A is the string β of the adjacent constituent B of A. Parsing is then performed on the "extended" string $\beta\alpha$, to test whether this string leads to some ambiguity. If the "extended string" is either not parse-able or is not ambiguous we conclude that the newly produced string α does not force ambiguities in the current state of computation of generating the final string.

For example suppose that an utterance with meaning 'Remove the folder by means of the system tools.' has to be produced. Furthermore, suppose that the partial string 'Remove the folder' has been generated using a rule 'vp $\rightarrow$ v, np, pp'. Now, the result of generating the pp is 'with the system tools'. In order to check whether this string is ambiguous 'the folder' is used as context and the string 'the folder with the systems tools' is parsed. This string is parse-able if a rule e.g., 'np $\rightarrow$ np, pp' exists. If it is parse-able then a source of ambiguity has been found, so that pp should be revised. If revision is not possible, then revision of the previous chosen vp should take place. However, if the rule 'vp $\rightarrow$ v, pp, np' had been chosen, and the currently produced string is "the folder", then the extended string to parse would be "with the system tools the folder". In this case, however, the string would not be parse-able. For the monitoring strategy this means, that at this point of computation, no statement of a possible ambiguity can be made, so the revision should not take place. In other words, the newly produced string "the folder" does not cause a relevant ambiguity in the current domain of locality spanned by the vp rule.

The proposed approach realizes a kind of *look-back* strategy, in the sense, that the monitor look backs to already produced substrings, in order to test whether a new string together with previous produced substrings causes ambiguity. For the method described so far, we actually have made a look-back of one adjacent constituent. In principle, however, it is possibly also to take into account the adjacent element of an adjacent element, leading to a look-back(n) strategy. The degree of look-back used, directly influences the degree of ambiguity we are going to consider. For example suppose we have the following grammar (s, np, vp are non-terminals, the other symbols are terminal elements):

1. s $\rightarrow$ np vp
2. vp $\rightarrow$ a b c d
3. vp $\rightarrow$ a np
4. np $\rightarrow$ x y
5. np $\rightarrow$ b c d

Assume that we have first chosen the vp rule 2., and the newly generated element is d (we assume a left-to-right scheduling). If we are following a look-back(1) strategy, then we have to parse the string 'cd'. This string, however, is not parse-able, so we do not try to revise it at that point. However, if we would use look-back(2), the string to parse would be 'bcd'. This string is parse-able, so there is the possibility of an ambiguity.

We are now going to describe how the look-back strategy informally described above can be integrated into the uniform algorithm developed in chapter 4 in order to perform the desired incremental monitoring strategy. Basically, we have to discuss the following questions:

1. How is the context determined and used for locating potential ambiguities?

2. How do we realize revision within the uniform algorithm?

The first question is concerned with the problem of determining context. This implies that we have to consider the possible different granulations the shape a context can have. For example, does it make sense to consider only one word as context or should it be better the string of complex constituents. This question will be considered after having introduced how revision should take place, because activation of revision is triggered after having determined a context, but it is possible to discuss revision by just assuming that context has already been determined.

Thus, the first question we want to consider, is the question how revision is realized within the uniform algorithm.

5.7.3 Performing Revision Within the Uniform Algorithm

It turns out that performing revision during generation of an utterance using the uniform algorithm is not that difficult as it might be at a first glance.

Recall that the uniform algorithm keeps track of partial (complete or incomplete) results using an agenda and a chart. The agenda is used to maintain all newly created items before they are added to the chart. The selection strategy used by the agenda determines in which order the items are added to the chart. Following a depth-first strategy, for instance, then only those items are considered that eventually can contribute to the complete generation of the first possible utterance. All remaining alternative items are only added to the chart in the case that additional paraphrases are requested, e.g., when all possible strings of a given semantic expression shall be computed.

If an item has been added to the chart, the different inference rules are applied which eventually creates new items which are then added to the agenda. But note that only those created items which have been added to the agenda will be considered during further computation. By means of this "built-in" mechanism revision can be performed as follows: Suppose that we have deduced a new passive item p. This means that we have computed a new partial string. If p is added to the chart, by means of passive completion it is checked whether p can reduce an active item a. Then, before a is actually be reduced using p it is checked whether p causes an ambiguity using an appropriate context.

Only if no ambiguity can be determined, the reduction of a is performed and the resulting new item is added to the agenda. On the other side, if an ambiguity is recognized, then reduction will not be performed, and as a consequence no new item is created. This implies for a, that reduction of its selected element will only be performed if there is another alternative for p available on the agenda (or items which lead to the computation of the alternative). However, this alternative item will automatically be added to the chart by the agenda at some later point. In some sense, this kind of processing means that the selected element has implicitly been marked, and the agenda will choose an alternative item which corresponds to a selection of an alternative rule.

If no alternative for p can be deduced (i.e., either no further alternative exists, or no unambiguous alternatives exist), then a will never be completed. However, this means that the agenda automatically will add an alternative item of a (if present) to the chart, which then might be combined with p. Note that this reduction would be performed by active-completion, and hence, would reuse results of previously made computations. If this is the case, the marker of p implicitly has been pushed one level up. Since, the whole process is performed recursively, it might be the case that markers are pushed implicitly up to the initial root node. However, in all cases, we can benefit from the results of previously made computations.

We will use our pp-attachment example at that place to clarify the strategy. We are assuming the following simple grammar:

1.	s	$\rightarrow$	np vp
2.	vp	$\rightarrow$	v np pp
3.	vp	$\rightarrow$	v pp np
4.	np	$\rightarrow$	det n
5.	np	$\rightarrow$	np pp
6.	pp	$\rightarrow$	prep np

We assume that these rules are added to the agenda according to the order in which they are specified in the grammar. Using a depth-first selection strategy for the agenda, rule 2. is processed before 3. At some point the pp is produced, and will be used by passive completion to reduce an instance of rule 2. However, before the pp of vp is reduced, the string of the np is used as context for checking whether the pp causes ambiguity. Therefore, we parse the string of np-pp, and actually detect an ambiguity. For the pp, however, we have no further alternatives available on the agenda, so rule 2. cannot be reduced completely, i.e, for that rule the inference rules cannot create items to put on the agenda. However, the agenda mechanism guarantees that rule 3. will be selected. Reducing rule 3. by means of active-completion will first use the pp for reduction, assumed without ambiguity problems. Next the np should be used for reduction. Before that, however, the string of pp-np is monitored, which however cannot be parsed, and hence no revision is necessary. Thus, rule 3. will be reduced by the np to give a completely reduced vp, which then is used for reduction of rule 1.

Based on the observations made above, we can adapt the uniform algorithm for performing revision in the following way, assuming that we already know how to detect ambiguities in the incremental mode (how this is actually be performed is given in the next section): Revision should only take place if there exists a passive item which can be used for reducing an active one. Thus, we only have to consider revision for the inference rules ACTIVE-COMPLETION, SCANNING, and PASSIVE-COMPLETION, whose indexed versions can be found in section 4.10 of chapter 4.

In all three cases we add a further conditional statement around the body of the for all loop, namely that the body should only be evaluated if revision is not requested. For example, the PASSIVE-COMPLETION rule is changed as follows (only the relevant parts are expressed explicitly):

(1)	**proc**	PASSIVE-COMPLETION (**in**: PL; **out**: Candidates?):
(2)		"as it is"
(3)	$\forall$ AL $\in S_{from(PL)}$;	Φ = UNIFY(SEL(AL),HEAD(PL)) and $\Phi \neq \bot$
(4)	**do**	**if** NOT(AND(MONITOR?,REVISION_P(Φ[AL],PL)))
(5)	**then do**	Red := Φ[AL-SEL(AL)];
		"and so on, as above"
	od fi	
	od;	

In the relevant part of the new code we have added a new line (4), which says that the next operations (i.e., putting a just reduced active item on the agenda) will only be performed if the monitor mode is switched on (which is done by using a global variable MONITOR?, whose boolean value indicates whether processing should be performed with or without incremental monitoring) and if no revision has taken place, which is determine by the predicate REVISION_P.[10]

In the same manner ACTIVE-COMPLETION and SCANNING are modified. The definition of REVISION_P is as follows:

```
revision_p(AL,PL):
   extended-string := get_context(AL,PL,n);
   if extended-string
    then
     parsed_result := parse(extended-string)
      if and(parsed_result,ambiguous(AL,parsed_result))
       then TRUE
       else FALSE
      fi
    else FALSE
    fi.
```

The contextual information is determined by means of the function GET_CONTEXT. If so, the parser is called with the extended-string, built inside GET_CONTEXT, using a look-back of n, which value is set globally. To be more precise, first a new string is computed on the basis of the context and the passive item's string, and then the parser is called. Only if the parser successfully obtained one or more readings and if the result is ambiguous should revision take place.

Note that the way the uniform algorithm maintains the agenda and the chart, the incremental method "simulates" marking and revision of generated derivation trees as is done explicitly by the non-incremental method. However, marking is done implicitly — it is just a side effect of the uniform algorithm by not creating items which could cause ambiguity problems. Furthermore, because monitoring is applied on intermediate results, it is actually performed incrementally.

5.7.4 Performing Ambiguity Checks within the Uniform Algorithm

We now turn our attention to the problem of testing whether a new partial produced string causes ambiguity or not. To solve this problem, we have to specify how an appropriate context is determined, how this context is used for parsing, and how the result of parsing is analysed with respect to its ambiguity.

[10] Using a globally set flag to trigger incremental monitoring can also be useful if the flag can be switched off in a kind of any-time mode. For example, if the overall system receives important time constraints and if it is possible to change the value of MONITOR? from true to false interactively, the remaining semantic expression would be generated without monitoring.

Determination of Context The basic assumption behind the use of contextual information during the incremental monitoring strategy is that it only makes sense to test whether a partial string, say α, is ambiguous with respect to a larger string which entails α. Such a larger string will be built by means of concatenation of α and some other already produced string, which we will call the *contextual string* of α.

Since revision will be performed before a passive item PL is used for reduction of an active item AL, this active item defines the domain of locality from which contextual information can be determined. Completion will be performed if PL and the selected element of AL can be unified. Therefore, only those elements of the body will be considered as possible contextual strings, that have already been deduced as subgoals of the active item.

Completion causes the removal of the completed elements from the body of a clause, so the elements of the body cannot be used directly. However, using the modified representation of derivation trees as already used in the non-incremental method, we are able to extract the corresponding strings of completed elements of an active lemma from the mother node of AL. Thus, we will determine the contextual string of PL on the basis of the derivation tree represented as part of the constraints of the head of the lemma of AL.

Note that the call of the incremental monitoring mechanism, i.e., the call of REVISION_P is performed in a completion rule before the new reduced item is computed but after unification of the passive item with the selected element of the active item has been done. This guarantees that monitoring is only performed on consistent structures. As a side effect of unification, the derivation tree of the passive item is unified into the derivation tree of the head of the lemma of the active item.

For example, assume that we have reduced the grammar rule $vp \leftarrow v, np, pp$ up to the point where we only need to complete the pp in order to complete the vp. The corresponding active item would be of form

$$\langle vp \leftarrow pp, 0, ea(0), ea(i) \rangle$$

At that point the derivation tree represented as part of the constraints of vp is (making use of useful abbreviations):

$$\begin{bmatrix} \texttt{rn} & \mathit{vp3} \\ \texttt{string} & \langle remove, the, folder \rangle\text{-}P \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \left\langle \begin{bmatrix} \texttt{rn} & \mathit{v5} \\ \texttt{string} & \langle remove \rangle\text{-}P1 \\ \texttt{sem} & \mathit{dots} \\ \texttt{dtrs} & \langle\ \rangle \end{bmatrix}, \begin{bmatrix} \texttt{rn} & \mathit{np3} \\ \texttt{string} & \langle the, folder \rangle\text{-}P2 \\ \texttt{sem} & \mathit{dots} \\ \texttt{dtrs} & \textit{``its tree''} \end{bmatrix}, Tree \right\rangle \end{bmatrix}$$

where the variable $Tree$ is a pointer to the derivation tree of the selected element pp, which is still un-instantiated.

After successful unification of a passive item pp with the selected element, the derivation tree looks as follows:

$$\begin{bmatrix} \texttt{rn} & \textit{vp3} \\ \texttt{string} & \langle remove, the, folder, with, the, tools\rangle\text{-}P \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \left\langle \begin{bmatrix} \texttt{rn} & \textit{v5} \\ \texttt{string} & \langle remove\rangle\text{-}\textit{P1} \\ \texttt{sem} & \textit{dots} \\ \texttt{dtrs} & \langle\,\rangle \end{bmatrix}, \begin{bmatrix} \texttt{rn} & \textit{np3} \\ \texttt{string} & \langle the, folder\rangle\text{-}\textit{P2} \\ \texttt{sem} & \textit{dots} \\ \texttt{dtrs} & \textit{``its tree''} \end{bmatrix}, \begin{bmatrix} \texttt{rn} & \textit{pp1} \\ \texttt{string} & \langle with, the, tools\rangle\text{-}P \\ \texttt{sem} & \textit{dots} \\ \texttt{dtrs} & \textit{``its tree''} \end{bmatrix} \right\rangle \end{bmatrix}$$

Now, we take this representation as the basis for determination of the contextual string of the pp's string "with the tools" making use of a *look-back* strategy as already informally described above.

Recall that the value of the DTRS feature is a sequence of the derivations trees of the corresponding elements of the body. In this context, we will call the value of the DTRS feature the *sequence of sisters* of the node represented by the clause's head element.[11]

Since we consider the sister nodes as totally ordered in a sequence, a look-back one strategy (written as look-back (1)) of the selected element, is just the choice of its left or right sister node. Thus, for the example above, we choose the node labelled $np3$. From this derivation tree we choose the value of the STRING feature as contextual string. Since, we assume that strings are represented as difference lists, it will be the case, that the string of the root node of the derivation tree of np already entails the string of *pp*. Thus, we can directly start parsing of this string, to test whether this string is ambiguous.

Note that in the above example we have implicitly assumed, that the elements of the body are processed in a left-to-right manner. Of course, in the case of generation this is not the general case. It might be possible, that for example, the *pp* is completed before the *np* is. In this case, we would have no (left) sister to be use-able as contextual string for the *pp*, because the derivation tree of the *np* still needs to be constructed, which means that the position of this derivation tree within the sequence of sisters is still occupied by an un-instantiated variable. If this is the case, we conclude that for the *pp* no statement about ambiguity can be made, and therefore, no revision should take place. After the *np* has been completed, monitoring for the *np* will eventually take place. But now, there is a choice point for the *np* either to choose its left or right sister as the base of contextual information, or both.

We can directly generalize the informal description of a look-back(1) strategy to a look-back(n) strategy, if we not only consider the left or right sister node of

[11] The notion SISTER can be defined on the basis of *dominance* as follows (see [Partee *et al.*, 1990]): A node x *dominates a node* y if there is a connected sequence of branches in the tree extending from x to y. If x and y are distinct, x dominates y, and there is no distinct node between x and y, x *immediately dominates* y. A node is said to be the *daughter* of the node immediately dominating it, and distinct nodes immediately dominated by the same node are called *sisters*.

the selected element as context but the sequence of the n left or right sisters of the selected element. In order to do this we have to consider the following cases:

- one of the n sisters is un-instantiated, and
- there are less then n possible sister nodes to the left or right of the selected element.

The first case means that there is a sister which derivation tree has still not been computed. This means that we cannot determine the whole contextual string corresponding to the n sisters, and we conclude that no contextual string exists. The second case means that the whole set of left or right sisters of the selected element can be used as contextual information by actually performing a look-back of less than n. In that case we use the corresponding contextual string spanned by the sisters and use it for the ambiguity check.

For a more readable definition of the look-back(n) strategy, we make use of the notation $subseq(i, j)$, which is a subsequence of elements ranging form i to j. For example, $subseq(3, 5)$ denotes the subsequence $\langle c, d, e \rangle$ of the sequence $\langle a, b, c, d, e, f, g \rangle$. Empty production will be handled so that if the sequence contains the name of an empty production we just skip this element. For example, if a and b are empty productions, then the sequences $\langle a, c, a, b, d, e \rangle$ and $\langle c, d, e \rangle$ are considered as being equal. The notation "the string of $subseq(i, j)$" means the string built by a left to right concatenation of the strings of the elements of the subsequence (modulo empty productions). We will say that a "$subseq(i, j)$" is instantiated if for each element of the subsequence its derivation tree is instantiated.

Thus, the look-back(n) strategy can be expressed as follows: Let $\langle d_1, \ldots, d_m \rangle$ be the sequence of sisters of the derivation tree of a rule and let d_i be the derivation tree of the "unified" selected element of the rule, and α its string. Let ll be the length of $subseq(1, i-1)$ and rl be the length of $subseq(i+1, m)$. If $n > ll$ then let n be ll, and analogously let n be rl, if $n > rl$. Then,

- if $subseq(i-n, i-1)$ is instantiated but not $subseq(i+1, i+n)$ then let β be the string of $subseq(i-n, i-1)$; let $\beta\alpha$ be the extended string;
- if $subseq(i+1, i+n)$ is instantiated but not $subseq(i-n, i-1)$ then let β be the string of $subseq(i+1, i+n)$; let $\alpha\beta$ be the extended string;
- if $subseq(i-n, i-1)$ and $subseq(i+1, i+n)$ are instantiated with strings β and γ respectively then let $\beta\alpha\gamma$ be the extended string;
- otherwise, no contextual string exists, which is indicated by the boolean value **false**.

This definition is used inside the function GET_CONTEXT (which is called inside REVISION_P, see above) which receives as input an active and a passive item and returns either an extended string or false. To be more precise, we have:

```
get_context(AL,PL,n):
 dtrs := get_dtrs(AL);
 lsisters := get_left_sisters(AL,label(PL));
 rsisters := get_right_sisters(AL,label(PL));

 ``apply look-back(n) on lsisters and rsisters;''

 if extended-string then
  return extended-string
  else
  return false.
```

We first extract the sisters of the derivation tree of the active item AL, i.e., the value of the path $\langle deriv, dtrs\rangle$ of the constraints of the active item's lemma's head. We then split this list into a left and right subsequence, where the passive item (which corresponds to the unified selected element of AL) serves as the splitting point. Next, we apply the look-back(n) strategy, and either return an extended string or **false**, if no such exists.

Check ambiguity Next we call the parser (i.e., we run our uniform algorithm in the parsing mode), whose task is to parse the extended string. If the extended string cannot be parsed, we conclude that no revision is necessary, and the call of REVISION_P terminates with **false**. However, if the parser returns one or more results (which corresponds to semantic readings of the extended string), we apply the ambiguity check performed inside the function AMBIGUOUS (see below). Only if a parsed result exists and the result is ambiguous, REVISION_P returns **true** which will cause revision of the new string spanned by the passive item.

The ambiguity check is performed as follows. First we delete all spurious ambiguities in the same way as in the non-incremental method, i.e., for a pair of derivation trees which have the same semantics we only retain one. After this operation we may have either only one reading or a set of readings. The latter case means that there are different possibilities to assign a meaning to the extended string, therefore revision for the new string should take place.

The former case is a bit more complicated. Although this case means that the extended string has been analysed as unambiguous (since we have obtained only one result), it might be the case that this reading is the same as that of the semantic expression of the active item's lemma. In this case, we have just detected a spurious ambiguity, and therefore revision should not take place. If on the other side, the semantic expression is not equal to that of the active item, we have found an possible ambiguity, and hence, revision should take place.

The following description of the function AMBIGUOUS summarizes the different cases:

```
ambiguous(ParsedResult,AL):
 ReducedResult := ``delete spurious ambiguities
                   in the same way as known from the non-incremental
                   version'';
 if card(ReducedResult) > 1
  then return true
 else
  if sem(ReducedResult) = sem(AL)
   then return false
  else true.
```

5.7.5 Using Shared Items during Incremental Monitoring

The main advantage of the incremental method using the uniform algorithm described so far is that we benefit from the use of the chart during the monitored generation strategy, because also in that case we can reuse previously made computations. Since revision is automatically performed by the agenda mechanism of the uniform algorithm (by not creating items for those structures where an ambiguity as been detected), the main effort we have to spend to realize monitoring is the parsing operation performed on extended strings. (The determination of the contextual string is not a time critical operation.) We now show how the incremental monitoring method can be made more efficient by making use of the *item sharing* approach described in 4.15.

Recall that in the item sharing approach passive items that have been computed in one direction can directly be used in the other. Following the method described in 4.15 the uniform tabular algorithm maintains different agendas, item sets and active items for the parsing and generation mode, but passive items are shared during both directions. The object-oriented realization of the item sharing approach allows the parser (i.e. the parsing mode of the uniform algorithm) to be chart-based even when it is called inside the generator. Thus, if the parser is called via monitoring it can reuse previously self-made results at any stage.

By use of the item sharing approach partial results (i.e., passive items) are continually made available for the other direction. However, this means that for both directions it is the case that one direction can reuse results from the other directions.

For example, for the interleaved parsing mode this means that it can reuse results computed through generation when making the ambiguity check. During this job, however, it can provide results for the generator of which the generator can make use. This means, that parsing results are used through generation and generation results are used through parsing in an interleaved mode.

5.7.6 Implementation

The incremental monitoring mechanism has fully been implemented on top of the item sharing approach and tested with several smallish grammars. Although we currently do not make use of preferences-based strategies (see also the final chapter) we have paid attention to be as flexible as possible with respect to this future extension. For example, if the Prolog-like interactive mode is activated also during parsing a user can choose which of the results computed during parsing should be ignored for the ambiguity check. Furthermore, it is possible to stop the parser after a first result accepted as most suitable for the user has been computed. Thus a user can interactively specify which reading she prefers.

Furthermore, it is possible to switch off the monitor interactively by just changing the Boolean value of the global variable MONITOR? which is used to trigger monitoring. Currently, this can be done in combination with the interactive mode. If the flag is switched off further generation is automatically continued without monitoring. Using this mechanism it is possible to simulate some kind of any-time behaviour of the incremental method.

5.7.7 Properties of the Incremental Method

The incremental monitoring method can be seen as an additional restriction to the uniform algorithm to keep track only those computed partial results which do not force ambiguities. Note that monitoring is only triggered by the completion rules and will only be performed on consistent structures. The effect of monitoring is that the uniform algorithm will only consider a subset of possible answers, namely those which are un-ambiguous. If no un-ambiguous string can be produced then the resulting set of answers is empty. However, if the algorithm finds an answer then it is correct. In this sense the monitor just further constraints the set of computable answers for a given semantic expression.

There are two parameters which influence the behaviour of the incremental monitoring strategy: the concrete value of n for the look-back strategy and the degree of the nodes of a derivation tree, which corresponds with the length of the right hand side of the rules. We will call this the *branching factor* of the grammar. The maximal possible degree of a node will be denoted as *maximal branching factor*, and corresponds to the rule with the largest number of right-hand side elements defined in a grammar.

Recall, that the variable n refers to the number of sisters of a selected element of an active item which have to be considered as context. Now, suppose we have chosen 1 as the value of n, i.e., we are following a look-back(1) strategy. Furthermore, assume we have two grammars G_1 and G_2 which are weakly equivalent, and where the maximal branching factor of G_1 is 2 and that of G_2 is some integer m greater than 2. Thus, G_1 defines binary structures and G_2 flat structures (compared wrt. G_1).

For G_1 a look-back(1) strategy means, that in each case where the incremental monitor mechanism is activated the extended string determined on the basis of the contextual string of some active item is identical with the whole string of the constituent defined by the active item. The reason is that when using a grammar defining binary structures, each rule has maximally two right hand side elements, say s_1 and s_2. Only when both have been deduced, monitoring will take place, because otherwise no contextual information would be available. Suppose, that s_1 is completed before s_2, then the string of s_1 will serve as the contextual string of s_2. The extended string is then the concatenation of both strings, but this means that it is just the string spanned by the whole item. This implies, that all possible ambiguities will be detected and that if the incremental monitor generates an utterance, then this utterance is unambiguous.

For G_2 a look-back(1) strategy means in general, that only a substring of the string defined by a constituent will be taken into account when building an extended string. But then, as we already showed in in section 5.7.2, it is possible, that not all possible ambiguities will be detected. As a consequence, this means that if the incremental monitor generates a string, this string need not necessarily be unambiguous.

Putting both together, we obtain a different result (wrt. the degree of ambiguity of a "monitored generated string") using the same value of n, but on grammars which only differ with respect to their maximal branching factor. Of course, if we want to make sure that our algorithm behaves in the same way for grammars with different maximal branching factor, i.e., if it is to guarantee that only unambiguous strings are generated, then we have to choose the maximal branching factor of the grammar as the value for n when performing the look-back strategy.

On the other side, if we have a grammar with a maximal branching factor greater than 2, say m, then following a look-back(n) strategy, with $n < m$, would mean that the incremental monitoring strategy would only consider those ambiguities which can be detected with the chosen look-back strategy. However, in that case, there is no guarantee that the resulting string is unambiguous. But note that the value chosen for n directly influences the size of the extended string. This means, that in general, a small value for n would mean less effort for the parser.

The discussion made above directly reveals the problem of determining the appropriate value for the look-back strategy. If we choose the maximal branching factor, then we obtain unambiguous strings (if they actually exist), for the price of high computational effort. On the other side, if we choose a small value for n we reduce the effort but will eventually not obtain an unambiguous string. Furthermore, it cannot be guaranteed that we actually have considered relevant ambiguities.

In order to compromise between computational effort and the degree of resolved ambiguities, we have to consider some additional criteria, which are used to decide whether an ambiguity check should be applied to a newly generated string.

Assumed we have such criterions they can easily be used during monitoring, such that during the call of GET_CONTEXT this information is used firstly to check whether for the passive item ambiguity should take place, and second on each sister "consumed" by the look-back strategy the test are applied. Only if the passive item and its sisters fulfill the conditions expressed by these criterions an extended string will eventually be delivered.

This provides the possibility to restrict the application of the monitoring strategy, for instance, on grammar specific information. For example, it would be possible to restrict monitoring only for maximal projections or only for those structures which are known to cause ambiguities (e.g., pp-modifiers, coordinations). In our implementation we have already build in mechanisms that can take into account such additional grammar specific information. However it is a matter of future investigation (primarily on the linguistic side) to achieve meaningful and realistic criterions.

5.8 Conclusion

In this chapter we have presented basic strategies for performing self-monitoring and revision. We presented a non-incremental and an incremental version. The non-incremental version and the paraphrasing algorithm is a generalization of [Neumann and van Noord, 1992] and [Neumann and van Noord, 1994]. The incremental monitoring algorithm is novel. This is not surprisingly, because the incremental monitoring method to be practical must be built on top of a uniform method that supports efficient integration of parsing and generation. The item sharing approach and the uniform tabular algorithm are in particular suited to obtain this behaviour. Indeed, one of the motivations behind the development of the uniform tabular algorithm has been for interleaving parsing and generation, i.e, it has been practically motivated (which is not that bad; good ideas often come from practical needs).

Limitations It should be clear that monitoring and revision 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 logical form 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.

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 [Finkler and Neumann, 1989; 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.

Chapter 6

Summary and Future Directions

6.1 Summary

We have developed a uniform computational model for natural language parsing and generation. It is based on a novel uniform tabular algorithm for parsing and generation from constraint-based grammars, and a new method of grammatical processing called item sharing. On the basis of these methods we have shown how an elegant but practical interleaving of parsing and generation is achieved by a novel incremental monitoring algorithm that is used during natural language production. Implementations of these methods exist and we have given details about the technical realization.

The new uniform tabular algorithm is a generalization of the Earley deduction method introduced by [Pereira and Warren, 1983]. Although uniformly defined the algorithm is fully driven by the structure of the actual input – a string for parsing and a semantic expression for generation. This task-oriented behaviour is obtained by means of a data-driven selection function (the element to process next is determined on the basis of the current portion of the input) and a data-driven uniform indexing technique. It is uniform in the sense that the same basic mechanism is used for parsing and generation, although parameterized with respect to the information used for indexing lemmas. More precisely, in the case of parsing, lemmas are indexed using string information and in the case of generation semantic information is used to access lemmas. The kind of index causes completed information to be placed in different state sets. Using this mechanism we can benefit from table-driven generation, similar to that of parsing. For example, using a semantics-oriented indexing mechanism during generation massive redundancies are avoided, because once a phrase is generated, we are able to use it in any position within the sentence.

Since the only relevant parameter our algorithm has with respect to parsing and generation is the difference in input structures, the basic differences between parsing and generation are simply the different input structures. This seems to be trivial; however, our approach is the first uniform algorithm that is able to adapt its behaviour dynamically to the data, achieving *a maximal degree of uniformity of parsing and generation.* None of the current uniform approaches exhibit such a degree of uniformity. Moreover – in some sense as a side-effect – we have shown that it is superior to the semantic-head driven generation algorithm developed by [Shieber *et al.*, 1990] which is currently the most prominent algorithm used for grammatical generation.

There is evidence that comprehension and generation are not just inverses, but that they are related to each other also at the processing level. For example, the human mechanism also involves some monitoring of the output and it is widely accepted that this is performed by making use of the comprehension mechanism. However, it has been an open question as to how such a behaviour can practically be realized in computer systems. We have paid serious attention to that problem, and we obtained as an answer that systematic pursuit of uniformity in natural language processing – as followed in this thesis – achieves the necessary preconditions for a practical interleaving of parsing and generation.

The specific results we have obtained are twofold. First, we have shown that the uniform tabular algorithm can straightforwardly be extended in order to share partial results in both directions. We have called this property *item sharing*, because items (i.e., the internal representation of partial results) computed in one direction are automatically made accessible for the other direction as well, results computed during parsing are usable during generation and vice versa. Second, we have specified an incremental monitoring mechanism in order to demonstrate how an interleaved approach can contribute to the solution of complex problems. The underlying mechanism used during monitoring can be denoted as an *incremental generate-parse-revise* strategy: substrings produced during generation are parsed to test whether they lead to ambiguities, detected ambiguities are handled by means of revision. This mechanism has been integrated into the uniform algorithm in an elegant and practical way.

In this thesis we have only considered self-monitoring and revision in depth at the grammatical level. However, by showing in detail how the uniform model contributes to the solution of this problem, we were able to demonstrate that uniformity is in fact of important practical relevance for natural language systems. By considering uniformity and interleaving of natural language parsing and generation under a strictly computational view we have broken new ground. In the next section we will discuss further important research directions for uniform processing.

6.2 Future Directions

In this section we discuss some important extensions of the uniform approach introduced in this thesis. We start by describing how Explanation-based Learning can be integrated with the uniform tabular algorithm to speed up processing. In [Neumann, 1994] we already reported on an implementation of such approach, however, only for the case of parsing. In the next section, we outline how this method can also be applied during generation.

In the section that follows we discuss the integration of preferences, processing of elliptical utterances, and fully incremental text processing. Finally, we discuss the relevance of the new uniform computational model under the perspective of cognitive processing.

6.2.1 Application of Explanation-Based Learning for Efficient Processing of Constraint-based Grammars

Explanation-based learning (EBL) is a technique through which an intelligent system can learn by observing examples. An EBL system derives justified generalizations from training instances on the basis of declarative background knowledge. As a method, EBL performs four different learning tasks: generalization, chunking, operationalization and justified analogy [Ellman, 1989]. Typically, the purpose of EBL is to produce a description of a concept that enables instances of that concept to be recognized efficiently [Minton *et al.*, 1989]. More fundamentally, EBL is a method for improving problem solving performance through experience. From this perspective, EBL is also of cognitive relevance, since it can be used to explain why humans tend to phrase ideas in the same way most of the time by adapting to a collection of idioms or prototypical constructions.

In [Neumann, 1994] I have described the application of EBL to efficient parsing of constraint-based grammars. The idea is to generalize the derivations of training instances created by normal parsing automatically and to use these generalized derivations (also called templates) during the run-time mode of the system. In the case that a template can be instantiated for a new input, no further grammatical analysis is necessary. The approach[1] is not restricted to the sentential level but can also be applied to arbitrary sub-sentential phrases, i.e., it is possible to handle substrings of an input by templates. Therefore, the EBL method can be interleaved straightforwardly with normal processing to get back flexibility that otherwise would be lost. In the paper mentioned above we have shown how this interleaving is obtained by using an agenda-based Earley style parser.

For those strings which can be completely processed using EBL we have achieved a speed up of factor thirty to fifty compared with the time required for full parsing

[1] The application of EBL for natural language processing just described is closely related to the method of Derivational Analogy (DA), developed by Carbonell to investigate analogical reasoning in the context of problem solving [Carbonell, 1983]. The DA technique solves a new problem by making use of a solution derivation that was generated while solving a previous problem. Solving new problems is performed by modifying previously obtained derivations. Because the derivations are justified, DA can be seen as a type of justified analogical reasoning.

with the large German HPSG grammar of the DISCO system [Uszkoreit *et al.*, 1994], to which the method has been integrated. However, because the parser can also be "initialized" with templates instantiated for substrings of an input, the speed is also enormously increased for input that cannot completely be covered by the EBL method.

Since we assume that the training phase is driven by a representative corpus the EBL method can also be used as the basis for sub-language approaches, for example, in order to train a system to use only those constructions which are most predictable in a specific domain. Thus, for this domain the system would have less coverage but would be able to process the sub-language very efficiently.

The basic idea of applying EBL in the context of constraint-based natural language grammars is due to [Rayner, 1988] and [Samuelsson and Rayner, 1991]. Samuelsson [1994] has further improved these methods by showing how EBL can be applied to a competence grammar and a representative training set of input sentences by extracting a learned grammar. The learned grammar is complied into LR parsing tables and a special LR parser enables very much faster parsing than the original one does. The important point of Samuelsson's approach is, that he *replaces* the competence grammar with the performance grammar. Thus, his approach could be seen as a kind of *corpus-based grammar compilation*. Hence, integration with normal processing is not supported.

Although our method shares some commonalities with that of [Samuelsson, 1994], there are substantial differences, most notably the interleaving with normal processing and its potential of "reversibility".

Up to now, we have only applied EBL to parsing. However, with our uniform tabular algorithm it is now also possible to apply the same method to *generation*, and to interleave EBL and normal generation in the same way as we do it for parsing. Note that after the generalization of a derivation tree, the resulting template is stored in a discrimination net using the generalized sequence of morphological analyses of the training instance as path expression. Therefore, templates can economically be stored and efficiently be retrieved. For generation, we would use a generalized description of the input semantics as index expression for the retrieval of the templates, which would have been computed in the same manner as during parsing.

Moreover, since we are using a reversible grammar, using the uniform algorithm with EBL means that the same set of templates could be accessed either from morphological or semantic information. This means that we also could extend the item sharing approach to yield a kind of *template sharing* approach.

At the DFKI we have begun to extend the EBL method described in [Neumann, 1994] exactly in that direction. The most basic problem for generation is to find a meaningful way of "generalizing semantic information". In the current implementation, for instance, we are abstracting away from some morphological features of each word, e.g., stem, number, tense.[2] However, it is yet an open question, which semantic information should best be generalized in order to obtain a similar behaviour.

A further important line of research will be to incorporate statistical methods in order to control the size and retrieval of the discrimination tree. A possible strategy is to attach preference values to the edges of the discrimination tree. These preference values can be determined automatically based on the frequency of successful retrieval and in dependence of the point of time of the last successful retrieval. The discrimination tree then can be seen as a self-organizing data-structure. In a similar way alternative templates can be organized to implement a kind of "preferred template first" strategy.

6.2.2 Further Important Directions

Using Preference-based Strategies. The integration of preference-based strategies into the new uniform model will be one of the main research direction in the near future.

We have mentioned several times the importance of preferences for natural language processing in the main chapters of this thesis (as well as for the EBL method) and we have been careful to avoid obstacles to this important future direction. The agenda mechanism of the uniform algorithm, for example, is already an important prerequisition for the incorporation of such strategies, since it allows processing of new items in any order. Also the architecture of the item sharing approach has been designed to support preference-based control.

The strategies described in [Uszkoreit, 1991] and [Barnett, 1994] seem to be suitable candidates for the new uniform environment. The work described in [Uszkoreit, 1991] is of importance since it focusses on the integration of preferences with the feature system of a constraint-based grammar as an appropriate means for obtaining plausible performance models. In [Barnett, 1994] a model is described that is able to handle specific preferences for parsing and generation, as well as shared preferences.

It is reasonable to assume that both strategies (even together) can be integrated into the new uniform model. If so, it would also be possible to realize a sort of preference-based monitoring strategy. We assume that the NLS in which the uniform model is integrated maintains different preference spaces for parsing and generation. Preference-based monitoring would then mean that the derivation of a produced utterance is directed so that it is consistent with respect to the assumed preferences of the interlocutor which have been used to direct parsing (clearly, this presumes that the NLS has as its disposal a user and discourse model). For example, if both prefer minimal–attachment of pp-modifiers then an utterance like "Remove the folder with

[2] Actually we can parameterize the method with respect to the information that should be generalized using a method similar to that of a restrictor. It has been shown that the amount of information generalized has a direct reflection on the space and time behaviour of the system.

the system tools" (with meaning "Remove the folder by means of the system tools") would cause no revision.

Processing of Elliptical Utterances. We will now outline how the processing of elliptical utterances can be performed using the item sharing approach.

For example, consider the following dialog between person A and the NLS B:

A: 'Peter is coming to the party tonight.'

B: 'Mary, too.'

Generation of the elliptical utterance 'Mary,too' can be performed by means of the item sharing method as follows. We assume that the structure of an item has an additional slot :producer which indicates whether an item has been constructed during parsing or generation.[3] If A's utterance has been analyzed we assume that a passive item for the VP 'is coming to the party tonight' is constructed. This item is now also available for generation. However, the value of the slot :producer is something like :parsing, since the parser has constructed this item. When the generator is going to use this item to complete its process, it can use this information through the elliptic generation process, for example, as a statement for suppressing uttering of this "re-used" string.

Clearly, this is a straightforward but naive approach. In general the process is more complicated e.g., if B's utterance were be 'Mary and John, too'. However, in combination with the EBL approach mentioned above it might be possible to handle these cases also.

In a similar way it would also be possible to use integrated generation in order to complete elliptic constructions during parsing, by making use of previously analayzed or produced constructions. Of course, this implies that the NLS has at its disposal a discourse and dialog model.

Note that this kind of processing can only take into account grammatical information. Thus it cannot be verified by the system that generated ellipses are not so brief as to be ambiguous or misleading. In order to solve these cases, it is an interesting issue to investigate whether and how this grammar-oriented method can be combined with knowledge-based methods, for instance, the one described in [Jameson and Wahlster, 1982] (see also section 5.3).

Fully Incremental Text Processing. Fully incremental text processing means that the input to an NLS is given in piecemeal fashion and that arbitrary changes to the input are handled, e.g., deletion or modification of the input. For example, in [Wirén and Rönnquist, 1993] a method for fully incremental parsing is proposed that is based on the Earley algorithm and the modification for its use in an incremental

[3]In our current implementation we have already built in this mechanism.

environment reported in [Wirén, 1992]. Clearly, if such modifications to the text are allowed, the overall process is non-monotonic.

In [Neumann and Finkler, 1990; Reithinger, 1991; Harbusch *et al.*, 1991] incremental generation systems are described that allow changes of the generated string, e.g., if an adjective modifier is to be inserted into an already produced nominal phrase this can be handled by means of some sort of repairing.[4]

The basic motivation behind fully incremental text processing is that it can be used to support highly interactive text-processing tasks, e.g., checking of grammar, spelling and style immediately and in real-time within a text-editing environment.

Most important from our uniform perspective is that [Wirén and Rönnquist, 1993] motivate a combination of incremental parsing and incremental generation based on a reversible grammar in order to explore highly interactive text-processing facilities, like structure-editing operations, propagation of grammatical changes, or on-line translation, in which the target-language text is generated in parallel with the source-language. However, they only mention this kind of integrated incremental parsing and generation approach as a further important research direction.

The incremental monitoring mechanism developed in this thesis already performs a kind of incremental parse-and-generate approach using a reversible grammar. By means of the item sharing approach we have demonstrated how such an interleaving of parsing and generation can be done efficiently. However, the incrementality is restricted since it does not allow for input given in a piecemeal fashion, and changes to the input are forbidden, too. The first restriction is not that problematic because our algorithm processes a given input on-line, i.e. it does not require an initialization of the chart. For the second case, it might be possible to adapt the update-mechanisms described in [Wirén, 1992] and [Wirén and Rönnquist, 1993] for the case of generation as well. Thus, the integrated approach to parsing and generation developed in this thesis seems also to be an important contribution for the investigation of fully incremental text processing.

6.2.3 Cognitive Processing

In cognitive linguistics an important aspect of developing processing strategies is that these strategies should be capable of modelling the humans behaviour in processing natural language. Therefore, an important task is to validate a proposed model by means of psychological experiments. However, this task – so far has I know – has only be done either for cognitive parsing or cognitive generation, but not under the assumption of uniform processing. Nevertheless, it is interesting enough to compare our approach with those approaches that model grammatical processing under a cognitive perspective. We first consider cognitive parsing. We will use the new work

[4]In [Finkler and Schauder, 1992] a more general discussion of the effects of incremental output on incremental generation can be found. In [Levelt, 1989] a detailed discussion of possible repairing strategies is given.

presented in [Hemforth, 1993] as our primarily basis for the discussion. We then discuss important aspects of cognitive generation.

It is important to emphasize at this point, that we do not claim cognitive plausibility of our uniform computational model, because we have not made psycholinguistic experiments that would support such a claim. Nevertheless, we have found out some interesting similarities with currently developed cognitive models so that it is legitimate to discuss the uniform model under a cognitive perspective.

Aspects of Cognitive Parsing

With respect to grammatical processing, it seems to be evident that humans do not compute in advance all possible readings of an utterance before some sort of disambiguation is performed in order to determine the "best" reading. Rather it seems to be the case that humans *prefer* a reading as most appropriate in a given situation. Therefore, most of the cognitive processing strategies follow some kind of *deterministic* strategy. These approaches can further be classified into those which are *strongly deterministic* and those which are *quasi-deterministic*. In quasi-deterministic models, a first reading of a sentence (or phrase) is constructed. If at a later point this preferred first reading is rejected because of syntactic, semantic or pragmatic reasons a second analysis is computed by means of *re-analysis*. In strongly deterministic models, no re-analysis is done.

In [Hemforth, 1993] several prominent cognitive models are discussed and empirically evaluated with respect to the following questions (under particular consideration of the German language):

- How do the models deal with structural ambiguities?
- How are word order preferences explained?
- When is lexical information taken into account?

Based on the empirical analysis presented in Hemforth's work, she deduces the following conditions for a cognitive plausible model of language processing:

- The human language process is *competence-based*; i.e., grammatical knowledge is fundamentally used for the analysis of language. Processing of grammatical knowledge is performed automatically and efficiently.
- Humans seem to process the grammatical structure of an utterance incrementally from left to right, i.e., words are integrated into the existing structure as fast as possible.
- Based on the experimental results presented in [Marslen-Wilson, 1976] , there is evidence that during the processing of a single utterance syntactic and semantic knowledge is already used in common to handle local disambiguations.

Together with the incremental nature of the parsing process Hemforth concludes that sign-based approaches seem to be most appropriate, especially HPSG ([Hemforth, 1993], page 180).

- In the case of structural ambiguities only one alternative is considered during further processing using preferences. If the chosen alternative has to be rejected the process of re-analysis is activated. Furthermore, the re-analysis is purposive and efficient.

Hemforth concludes that the only strategy that is consistent with the empirical evidence she had obtained through her experiments is the mixed bottom-up/top-down left-corner strategy of the Earley algorithm (see [Hemforth, 1993], page 176).[5]

Purely data-driven (*bottom-up*) and goal-directed (*top-down*) strategies are not consistent with the empirical results provided in Hemforth's work. For bottom-up strategies there is no prediction possible about items of the next input structure, although experiments have shown that humans actually use such predictions for disambiguation. Next, for right-branching structures no incremental processing is possible, because a new constituent can only be built if all parts are completely determined. Therefore the necessary amount of memory of keeping partial results is very high.

A *top-down* strategy allows the integration of new items into the sentence structure immediately, and each lexical item can be predicted on the basis of knowledge constructed so far. Therefore, disambiguation of lexical ambiguities as well the processing of right-branching structures can be made efficiently. However, simple top-down backtracking strategies have the well-known left-recursive problems and the use of backtracking would not explain why the re-analysis process is so efficient.

Hemforth's results support the uniform architecture as follows. First, we also follow the Earley strategy. Processing is performed left-to-right if the grammar defines string concatenation in that way. The algorithm is performed as an on-line strategy, which means that words of a string are consumed next by next. However, the incremental behaviour is restricted, because we do not pass the words of a string one by one as input to the uniform tabular algorithm (see also the notes about fully incremental text processing made in the previous section).

The uniform algorithm is directed by an agenda, and we have shown that this allows for depth-first and even preference-based strategies, although the latter has not yet been used. The underlying grammars used are sign-based. Lexical access is done as early as possible by taking into account a huge number of top-down predictions. In summary, our uniform algorithm reflects the requirements worked out in Hemforth's thesis and actually can be used as a basis for cognitive models – at least for parsing.

[5] [Johnson-Laird, 1983] has also advocated Early's parsing algorithm as the most plausible cognitive one, but he has considered less empirical data ([Hemforth, 1993], page 176).

Aspects of Cognitive Generation

For the case of cognitive generation the picture is unfortunately not that clear, because cognitive generation strategies have only recently been given significant consideration. However, from the work presented in [Kempen and Hoenkamp, 1987], [DeSmedt and Kempen, 1987], [Levelt, 1989], and [Pechmann, 1989], it seems to be evident that generation is also performed incrementally. Furthermore, all these approaches also assume that grammatical generation is competence-based, i.e., grammatical knowledge is used for the production of sentences. Moreover, there is psychological evidence that the same grammar is used for performing parsing and generation – as we have already outlined in the first chapter of this thesis.

In all of the work cited above it is assumed that the input is of semantic nature given as a functor/argument tree [Levelt, 1989]. However, it is assumed that the parts of the meaning of an utterance are passed in a piecemeal fashion. Since it is now possible that arguments can arrive before their functors or vice versa it is assumed that the grammatical structure is built up in a mixed bottom-up/top-down way (see [Kempen and Hoenkamp, 1987] and [DeSmedt and Kempen, 1987]).

Furthermore, for reasons of parsimony it is assumed that generation is also performed in a deterministic way [Kempen and Hoenkamp, 1987], i.e., not all grammatical possibilities for an utterance to be produced are explored simultaneously. However, because of the fact that humans are able to monitor what they have heard or said, it is assumed that in some situations some sort of re-synthesis (or revision) will take place [Levelt, 1989]. However, how these mechanisms are actually be performed has not been specified computationally. Thus, it seems to be an open question whether in the case of cognitive generation chart-based or backtracking strategies should be considered when performing the revision process.

We have shown in this thesis that methods for performing self-monitoring and revision during generation are suitably realized by means of a tight integration of parsing and generation. As we have made clear, such an approach can be made efficiently using the uniform tabular algorithm in combination with the item sharing approach. On the top of these methods we have developed a chart-based incremental monitoring strategy. The results of this thesis suggest that revision too benefits from the use of a tabular based strategy, since it can be performed purposively and efficiently.

In summary, there seem to be some evidence that our uniform computational model conveys some cognitive plausibility, and it would be of great interest whether there actually exists some empirical data that could support this assumption. In this thesis we have laid out important theoretical and practical foundations for achieving competence-based performance language models. We are strongly convinced that our work will help to realize "natural" natural language systems and we have outlined how our approach can valuable contribute to important new lines of future research directions.

Appendix A

A Sample Grammar

In this appendix we define the grammar that we are going to use to demonstrate the uniform tabular algorithm. It is basically the same grammar (with minor changes) as the one introduced in [VanNoord, 1993] to demonstrate his head-driven bottom-up algorithms. In the rules we already make use of the more readable notation introduced in chapter 3. The rules will be enumerated using the index (r_i). In the text, we basically refer to the individual rules using this index.

First, the grammar rules.

$$(r_1)\quad sign\left(\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & \mathit{Tail}\\ \texttt{sem:} & \mathit{Sem}\\ \texttt{lex:} & no\\ \texttt{v2:} & V\\ \texttt{phon:} & P_0\text{-}P\\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & vp\text{-}sc\\ \textsc{dtrs} & \langle D_1, D_2\rangle\end{bmatrix}\end{bmatrix}\right) \longleftarrow$$

$$sign\left({}_{Arg}\begin{bmatrix}\texttt{v2:} & no\\ \texttt{phon:} & P_0\text{-}P_1\\ \textsc{deriv} & D_1\end{bmatrix}\right), sign\left(\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & \langle Arg|Tail\rangle\\ \texttt{sem:} & \mathit{Sem}\\ \texttt{v2:} & V\\ \texttt{phon:} & P_1\text{-}P\\ \textsc{deriv} & D_2\end{bmatrix}\right)$$

$$
(r_2)\quad sign\left(\begin{bmatrix}\texttt{cat:} & \textit{root}\\ \texttt{sem:} & \textit{Sem}\\ \texttt{phon:} & P_0\text{-}P\\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & \textit{root}\\ \textsc{dtrs} & \langle D_1, D_2\rangle\end{bmatrix}\end{bmatrix}\right) \longleftarrow
$$

$$
sign\left(\begin{bmatrix}\texttt{cat:} & \textit{adv}\\ \texttt{sc:} & \langle Q\rangle\\ \texttt{sem:} & \textit{Sem}\\ \texttt{phon:} & P_0\text{-}P_1\\ \textsc{deriv} & D_1\end{bmatrix}\right), sign\left({}_{Q}\begin{bmatrix}\texttt{cat:} & q\\ \texttt{phon:} & P_1\text{-}P\\ \textsc{deriv} & D_2\end{bmatrix}\right)
$$

$$
(r_3)\quad sign\left(\begin{bmatrix}\texttt{cat:} & q\\ \texttt{sem:} & \textit{Sem}\\ \texttt{phon:} & P_0\text{-}P\\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & q\\ \textsc{dtrs} & \langle D_1, D_2\rangle\end{bmatrix}\end{bmatrix}\right) \longleftarrow
$$

$$
sign\left({}_{V}\begin{bmatrix}\texttt{cat:} & \textit{vp}\\ \texttt{lex:} & \textit{yes}\\ \texttt{phon:} & P_0\text{-}P_1\\ \textsc{deriv} & D_1\end{bmatrix}\right), sign\left(\begin{bmatrix}\texttt{cat:} & \textit{vp}\\ \texttt{sc:} & \langle\ \rangle\\ \texttt{sem:} & \textit{Sem}\\ \texttt{v2:} & V\\ \texttt{phon:} & P_1\text{-}P\\ \textsc{deriv} & D_2\end{bmatrix}\right)
$$

$$
(r_4)\quad sign\left(\begin{bmatrix}\texttt{cat:} & \textit{comp}\\ \texttt{sem:} & \textit{Sem}\\ \texttt{sc:} & \langle\ \rangle\\ \texttt{phon:} & P_0\text{-}P\\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & \textit{comp}\\ \textsc{dtrs} & \langle D_1, D_2\rangle\end{bmatrix}\end{bmatrix}\right) \longleftarrow
$$

$$
sign\left(\begin{bmatrix}\texttt{cat:} & \textit{comp}\\ \texttt{sem:} & \textit{Sem}\\ \texttt{sc:} & \langle \textit{Arg}\rangle\\ \texttt{phon:} & P_0\text{-}P_1\\ \textsc{deriv} & D_1\end{bmatrix}\right), sign\left({}_{\textit{Arg}}\begin{bmatrix}\texttt{phon:} & P_1\text{-}P\\ \textsc{deriv} & D_2\end{bmatrix}\right)
$$

$$(r_5)\quad sign\left(\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & Sc\\ \texttt{sem:} & Sem\\ \texttt{lex:} & no\\ \texttt{v2:} & V\\ \texttt{phon:} & P_0\text{-}P\\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & vp\text{-}adv\\ \textsc{dtrs} & \langle D_1, D_2\rangle\end{bmatrix}\end{bmatrix}\right) \longleftarrow$$

$$sign\left(\begin{bmatrix}\texttt{cat:} & adv\\ \texttt{sem:} & Sem\\ \texttt{sc:} & \langle Vp\rangle\\ \texttt{phon:} & P_0\text{-}P_1\\ \textsc{deriv} & D_1\end{bmatrix}\right), sign\left(Vp\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & Sc\\ \texttt{v2:} & V\\ \texttt{phon:} & P_1\text{-}P\\ \textsc{deriv} & D_2\end{bmatrix}\right)$$

$$(r_6)\quad sign\left(\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & Sc\\ \texttt{sem:} & Sem\\ \texttt{lex:} & no\\ \texttt{v2:} & \begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sc:} & Sc\\ \texttt{sem:} & Sem\end{bmatrix}\\ \texttt{phon:} & P\text{-}P\\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & vp\text{-}empty\\ \textsc{dtrs} & \langle\ \rangle\end{bmatrix}\end{bmatrix}\right)$$

Next, the lexical entries.

$$(r_7)\quad sign\left(\begin{bmatrix}\texttt{cat:} & comp\\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & unary\\ \texttt{pred:} & weil\\ \texttt{arg:} & Sem\end{bmatrix}\\ \texttt{sc:} & \left\langle\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sem:} & Sem\\ \texttt{sc:} & \langle\ \rangle\\ \texttt{v2:} & no-v2\end{bmatrix}\right\rangle\\ \texttt{phon:} & \langle weil|T\rangle\text{-}T\\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & weil\\ \textsc{dtrs} & \langle\ \rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_8)\quad sign\left(\begin{bmatrix}\texttt{cat:} & adv\\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & unary\\ \texttt{mod:} & heute\\ \texttt{arg:} & Sem\end{bmatrix}\\ \texttt{sc:} & \left\langle\begin{bmatrix}\texttt{sem:} & Sem\end{bmatrix}\right\rangle\\ \texttt{phon:} & \langle heute|T\rangle\text{-}T\\ \text{DERIV} & \begin{bmatrix}\texttt{rulename} & heute\\ \text{DTRS} & \langle\ \rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_9)\quad sign\left(\begin{bmatrix}\texttt{cat:} & np\\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & nullary\\ \texttt{pred:} & Peter\end{bmatrix}\\ \texttt{phon:} & \langle Peter|T\rangle\text{-}T\\ \text{DERIV} & \begin{bmatrix}\texttt{rulename} & peter\\ \text{DTRS} & \langle\ \rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_{10})\quad sign\left(\begin{bmatrix}\texttt{cat:} & np\\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & nullary\\ \texttt{pred:} & l\ddot{u}gen\end{bmatrix}\\ \texttt{phon:} & \langle L\ddot{u}gen|T\rangle\text{-}T\\ \text{DERIV} & \begin{bmatrix}\texttt{rulename} & luegen\\ \text{DTRS} & \langle\ \rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_{11})\quad sign\left(\begin{bmatrix}\texttt{cat:} & vp\\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & unary\\ \texttt{pred:} & schlafen\\ \texttt{arg:} & Sem\end{bmatrix}\\ \texttt{sc:} & \left\langle\begin{bmatrix}\texttt{cat:} & np\\ \texttt{sem:} & Sem\end{bmatrix}\right\rangle\\ \texttt{lex:} & yes\\ \texttt{v2:} & no-v2\\ \texttt{phon:} & \langle schl\ddot{a}ft|T\rangle\text{-}T\\ \text{DERIV} & \begin{bmatrix}\texttt{rulename} & schlafen\\ \text{DTRS} & \langle\ \rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_{12})\quad sign\left(\begin{bmatrix} \texttt{cat:} & vp \\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & binary\\ \texttt{pred:} & \textit{erzählen} \\ \texttt{arg1:} & Ag \\ \texttt{arg2:} & Th\end{bmatrix} \\ \texttt{sc:} & \left\langle \begin{bmatrix}\texttt{cat:} & np\\ \texttt{sem:} & Th\end{bmatrix}, \begin{bmatrix}\texttt{cat:} & np \\ \texttt{sem:} & Ag\end{bmatrix}\right\rangle \\ \texttt{lex:} & yes \\ \texttt{v2:} & no - v2 \\ \texttt{phon:} & \langle \textit{erzählt}|T\rangle\text{-}T \\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & erzaehlen \\ \textsc{dtrs} & \langle\,\rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_{13})\quad sign\left(\begin{bmatrix} \texttt{cat:} & vp \\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & unary\\ \texttt{pred:} & nickerchenmachen \\ \texttt{arg:} & Exp\end{bmatrix} \\ \texttt{sc:} & \left\langle \begin{bmatrix}\texttt{cat:} & np\\ \texttt{sem:} & Exp\end{bmatrix}, \begin{bmatrix}\texttt{cat:} & np \\ \texttt{sem:} & \begin{bmatrix}\texttt{pred:} & nickerchen\end{bmatrix}\end{bmatrix}\right\rangle \\ \texttt{lex:} & yes \\ \texttt{v2:} & no - v2 \\ \texttt{phon:} & \langle macht|T\rangle\text{-}T \\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & idiom1 \\ \textsc{dtrs} & \langle\,\rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_{14})\quad sign\left(\begin{bmatrix} \texttt{cat:} & np \\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & nullary\\ \texttt{pred:} & nickerchen\end{bmatrix} \\ \texttt{phon:} & \langle ein,\ nickerchen|T\rangle\text{-}T \\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & nickerchen \\ \textsc{dtrs} & \langle\,\rangle\end{bmatrix}\end{bmatrix}\right)$$

$$(r_{15})\quad sign\left(\begin{bmatrix} \texttt{cat:} & adv \\ \texttt{sem:} & \begin{bmatrix}\texttt{type:} & unary\\ \texttt{mod:} & gerne \\ \texttt{arg:} & Sem\end{bmatrix} \\ \texttt{sc:} & \left\langle \begin{bmatrix}\texttt{sem:} & Sem\end{bmatrix}\right\rangle \\ \texttt{phon:} & \langle gerne|T\rangle\text{-}T \\ \textsc{deriv} & \begin{bmatrix}\texttt{rulename} & gerne \\ \textsc{dtrs} & \langle\,\rangle\end{bmatrix}\end{bmatrix}\right)$$

Bibliography

[Aït-Kaci *et al.*, 1994] Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(1–2):263–283, January 1994.

[Allgayer *et al.*, 1989] J. Allgayer, K. Harbusch, A. Kobsa, C. Reddig, N. Reithinger, and D. Schmauks. Xtra: A natural–language access system to expert systems. *Int. J. Man–Machine Studies*, 31:161–195, 1989.

[Alshawi and Pulman, 1992] H. Alshawi and S. G. Pulman. Ellipsis, comparatives, and generation. In H. Alshawi, editor, *The Core Language Engine*, pages 251–276. MIT Press, Cambridge, MA, 1992.

[Alshawi, 1992] H. Alshawi, editor. *The Core Language Engine*. ACL-MIT Press Series in Natural Language Processing. MIT Press, Cambridge MA., 1992.

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

[Appelt, 1987] D. E. Appelt. Bidirectional grammars and the design of natural language generation systems. In Y. Wilks, editor, *Theoretical Issues in Natural Language Processing-3*, pages 185–191. Hillsdale, N.J.: Erlbaum, 1987.

[Backofen and Smolka, 1993] Rolf Backofen and Gert Smolka. A complete and recursive feature theory. In *Proc. of the 31 th ACL*, pages 193–200, Columbus, Ohio, 1993. acl. Full version has appeared as Research Report RR-92-30, DFKI, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany, and will appear in Theoretical Computer Science.

[Backofen and Weyers, 1993] R. Backofen and C. Weyers. *UDiNe*—A Feature Constraint Solver with Distributed Disjunction and Classical Negation. Technical report, DFKI, Saarbrücken, Germany, 1993. Forthcoming.

[Barnett, 1994] J. Barnett. Bi-directional preferences. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 201–234. Kluwer, 1994.

[Bateman *et al.*, 1992] J. A. Bateman, M. Emele, and S. Momma. The nondirectional representation of systemic functional grammar and semantics as typed feature structure. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, Nantes, 1992.

[Berg, 1986] T. Berg. The problems of language control: Editing, monitoring and feedback. *Psychological Research*, 48:133–144, 1986.

[Block, 1991] H. U. Block. Compiling trace & unification grammar for parsing and generation. In *Proceedings of the ACL Workshop Reversible Grammar in Natural Language Processing*, Berkeley, 1991.

[Block, 1994] Hans-Ulrich Block. Compiling Trace & Unification Grammar. In T. Strzalkowski, editor, *Reversible Grammar in Natural language Processing*, pages 155–174. Kluwer Academic Press, London, 1994.

[Bresnan, 1982] J. Bresnan, editor. *The Mental Representation of Grammatical Relations.* MIT Press, 1982.

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

[Calder *et al.*, 1989] J. Calder, M. Reape, and H. Zeevat. An algorithm for generation in unification categorial grammar. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 233–240, Manchester, 1989.

[Carbonell, 1983] J. G. Carbonell. Derivational analogy and its role in problem solving. In *AAAI-83*, pages 64–69, Washington, DC, 1983.

[Carpenter, 1992] B. Carpenter. *The logic of typed feature structures with application to unification grammars, logic programms and constraint resolution.* Cambridge University Press, Cambridge, 1992.

[Chomsky, 1986] N. Chomsky. *Knowledge of Language: Its Nature, Origin and Use.* New York, Praeger, 1986.

[Dale *et al.*, 1990] R. Dale, C. Mellish, and M. Zock. *Current Research in Natural Language Generation.* Academic Press, London, 1990.

[Dale, 1990] R. 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.

[Danlos, 1987] L. Danlos. *The Linguistic Basis of Text Generation.* Cambridge University Press, Cambridge, MA, 1987.

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

[Dörre, 1993] J. Dörre. *Feature-Logik und Semiunifikation*. PhD thesis, Universität Stuttgart, Stuttgart, Germany, 1993.

[Dymetman and Isabelle, 1988] M. Dymetman and P. Isabelle. Reversible logic grammars for machine translation. In *Proceedings of the Second International Conference on Theoretical and Methodological issues in Machine Translation of Natural Languages*, Pittsburgh, 1988.

[Dymetman *et al.*, 1990] M. Dymetman, P. Isabelle, and F. Perrault. A symmetrical approach to parsing and generation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, pages 90–96, Helsinki, 1990.

[Dymetman, 1991] M. Dymetman. Inherently reversible grammars, logic programming and computability. In *Proceedings of the ACL Workshop Reversible Grammar in Natural Language Processing*, Berkeley, 1991.

[Dymetman, 1994] M. Dymetman. Inherently reversible grammars. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 33–57. Kluwer, 1994.

[Earley, 1970] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[Ellman, 1989] T. Ellman. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys*, 21(2):163–221, 1989.

[Emele and Zajac, 1990] M. C. Emele and R. Zajac. Typed unification grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, pages 293–298, Helsinki, 1990.

[Erbach, 1991] G. Erbach. An environment for experimentation with parsing strategies. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 931–936, Sydney, Australia, 1991.

[Fedder, 1991] L. Fedder. Syntactic choice in language generation. In *Proceedings of the ACL Workshop on Reversible Grammar in Natural Language Processing*, pages 45–52, Berkeley, 1991.

[Fenstad *et al.*, 1987] J. E. Fenstad, P. K. Halvorsen, T. Langholm, and J. van Benthem. *Situations, Language and Logic*. Reidel, Dordrecht, 1987.

[Finkler and Neumann, 1989] W. Finkler and G. 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.

[Finkler and Schauder, 1992] W. Finkler and A. Schauder. Effects of incremental output on incremental natural language generation. In *Proceedings of the Tenth European Conference on Artifical Intelligence*, pages 505–507, Vienna, Austria, 1992.

[Fodor and Frazier, 1980] J. D. Fodor and L. Frazier. Is the human sentence parsing mechanism an atn? *Cognition*, 1980.

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

[Frazier, 1982] L. 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.

[Garrett, 1982] M. 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.

[Genesereth and Nilsson, 1987] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence.* Kaufmann, Los Altos, CA, 1987.

[Gerdemann, 1991] D. D. Gerdemann. *Parsing and Generation of Unification Grammars.* PhD thesis, University of Illinois, Cognitive Science, Technical Report CS-91-06, 1991.

[Görz, 1988] G. Görz. *Strukturanalyse natürlicher Sprache. Ein Verarbeitungsmodell zum maschinellen Verstehen gesprochener und geschriebener Sprache.* Addison-Wesley, Bonn - Reading, Massachusetts - Menlo Park, California, 1988.

[Haas, 1989] A. Haas. A parsing algorithm for unification grammars. *Computational Linguistics*, 15(4):219–232, 1989.

[Harbusch *et al.*, 1991] K. Harbusch, W. Finkler, and A. Schauder. Incremental syntax generation with tree adjoining grammars. Technical report, DFKI Saarbrücken, 1991. RR-91-25.

[Hemforth, 1993] B. Hemforth. *Kognitives Parsing: Repräsentation und Verarbeitung sprachlichen Wissens.* Infix, Sankt Augustin, 1993.

[Hoeppner *et al.*, 1983] W. Hoeppner, T. Christaller, H. Marburger, K. Morik, B. Nebel, M. 'Leary, and W. Wahlster. Beyond domain-independence: Experience with the development of a german language access system to higly diverse background systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 643–649, Karlsruhe, 1983.

[Höhfeld and Smolka, 1988] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical Report Technical Report No. 53,LILOG IBM, Stuttgart, 1988.

[Horacek, 1990] H. 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, 1990.

[Hovy, 1987] E. H. Hovy. *Generating Natural Language under Pragmatic Constraints.* PhD thesis, Yale University, 1987.

[ICSLP, 1992] *International Conference on Spoken Language Processing*, Alberta,Canada, 1992.

[Jacobs, 1988] P. S. Jacobs. Achieving bidirectionality. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, pages 267–274, Budapest, 1988.

[Jaffar and Lassez, 1987] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.

[Jameson and Wahlster, 1982] A. Jameson and W. Wahlster. User modelling in anaphora generation: Ellipsis and definite description. In *Proceedings of teh 1982 European Conference on Artificial Intelligence*, pages 222–227, Orsay, France, 1982.

[Johnson-Laird, 1983] P. N. Johnson-Laird. *Mental Models.* Harvard University Press, Cambridge, MA, 1983.

[Johnson, 1993] M. Johnson. Memoization in constraint logic programming. Department of Cognitive Science, Brown University, 1993. manuscript.

[Joshi, 1987] A. K. Joshi. Generation – a new frontier of natural language processing? In Y. Wilks, editor, *Theoretical Issues in Natural Language Processing-3*, pages 181–184. Hillsdale, N.J.: Erlbaum, 1987.

[Kay, 1986] M. Kay. Algorithm schemata and data structures in syntactic processing. In B. J. Grosz, K. Sparck Jones, and B. L. Webber, editors, *Natural Language Processing*, pages 35–70. Kaufmann, Los Altos, CA, 1986.

[Kay, 1989] M. Kay. Head-driven parsing. In *Proceedings of Workshop on Parsing Technologies*, pages 52–62, Pittsburgh, 1989.

[Keene, 1989] S. E. Keene. *Object-Oriented Programming in COMMON LISP: A Programmer's Guide to CLOS*. Addison-Wesley, Reading, MA, 1989.

[Kempen and Hoenkamp, 1987] G. Kempen and E. Hoenkamp. An incremental procedural grammar for sentence formulation. *Cognitive Science*, 11:201–258, 1987.

[Kempen, 1989] G. Kempen. Language generation systems. In I. S. Batori, W. Lenders, and W. Putschke, editors, *Computational Linguistics - Computerlinguistik*, pages 471–480. de Gruyter, Berlin, 1989.

[Kiefer and Fettig, 1994] B. Kiefer and T. Fettig. Fegramed an interactive graphics editor for feature structure. DFKI Document D-94-XX, DFKI GmbH, Kaiserslautern und Saarbrücken, Germany, 1994.

[Koskenniemi, 1984] K. Koskenniemi. A general computational model for word-form recognition and production. In *Proceedings of the 10th International Conference on Computational Linguistics and the 22nd Annual Meeting of the Association for Computational Linguistics (COLING)*, pages 178–181, Standford, 1984.

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

[Lloyd, 1987] J. .W. Lloyd. *Foundations of Logic Programming*. Symbol Computation, Springer, Berlin, New York, 1987.

[Mann, 1987] W. C. Mann. What is special about natural language generation research? In Y. Wilks, editor, *Theoretical Issues in Natural Language Processing-3*, pages 206–210. Hillsdale, N.J.: Erlbaum, 1987.

[Marslen-Wilson, 1976] W. D. Marslen-Wilson. Linguistic descriptions and psychological assumptions in the study of sentence perception. In R. .J.Wales and E. C. T.Walker, editors, *New Approaches to Language Mechanisms*. Amsterdam: North Holland, 1976.

[Martinovic and Strzalkowski, 1992] M. Martinovic and T. Strzalkowski. Comparing Two Grammar-Based Generation-Algorithms: A Case Study. In *30th Annual Meeting of the Association for Computational Linguistics*, pages 81–88, 1992.

[Matsumoto *et al.*, 1983] Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa. Bup: A bottom-up parser embedded in prolog. *New Generation Computing*, 1:145–158, 1983.

[McDonald *et al.*, 1987] D. D. McDonald, M. W. Meteer, and J. D. Pustejovsky. Factors contributing to efficiency in natural language generation. In K. Kempen, editor, *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, pages 159–182. Martinus Nijhoff, Dordrecht, 1987.

[McDonald, 1983] D. 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.

[McDonald, 1986] D. D. McDonald. Description directed control: Its implications for natural language generation. In B. J. Grosz, K. Sparck Jones, and B. L. Webber, editors, *Natural Language Processing*, pages 519–537. Kaufmann, Los Altos, CA, 1986.

[McDonald, 1987] D. D. McDonald. No better, but no worse, than people. In Y. Wilks, editor, *Theoretical Issues in Natural Language Processing-3*, pages 200–205. Hillsdale, N.J.: Erlbaum, 1987.

[McKeown *et al.*, 1990] K. R. McKeown, M. Elhadad, Y. Fukomoto, J. Lim, C. Lombardi, J. Robin, and F. 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.

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

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

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

[Minton *et al.*, 1989] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R.Kuokka, O. Etzioni, and Y.Gil. Explanation-based learning: A problem solving perspective. *Artifical Intelligence*, 40:63–118, 1989.

[Nerbonne, 1992] J. Nerbonne. Constraint-based semantics. Research Report RR-92-18, DFKI GmbH, Kaiserslautern und Saarbrücken, BR Deutschland, 1992.

[Netter, 1992] K. Netter. On non-head non-movement. In G. Görz, editor, *Konvens 92: 1. Konferenz "Verarbeitung natürlicher Sprache"*, pages 218–227. Springer, Berlin, Heidelberg, 1992.

[Neumann and Finkler, 1990] G. Neumann and W. 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.

[Neumann and van Noord, 1992] G. Neumann and G. van Noord. Self-monitoring with reversible grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pages 700–706, Nantes, 1992.

[Neumann and van Noord, 1994] G. Neumann and G. van Noord. Reversibility and self-monitoring in natural language generation. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 59–96. Kluwer, 1994.

[Neumann, 1991a] G. Neumann. A bidirectional model for natural language processing. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 245–250, Berlin, 1991.

[Neumann, 1991b] G. 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.

[Neumann, 1993] G. Neumann. Design principles of the disco system. In *Proceedings of the TWLT 5*, Twente, Netherlands, 1993.

[Neumann, 1994] G. Neumann. Application of explanation-based learning for efficient processing of constraint-based grammars. In *Proceedings of the Tenth IEEE Conference on Artifical Intelligence for Applications*, pages 208–215, San Antonio, Texas, March 1994.

[Norvig, 1992] P. Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp.* Kaufmann, San Mateo, CA, 1992.

[Paris *et al.*, 1991] C. L. Paris, W. R. Swartout, and W. C. Mann. *Natural Language Generation in Artificial Intelligence and Computational Linguistics.* Kluwer Academic Press, 1991.

[Partee *et al.*, 1990] B. H. Partee, A. ter Meulen, and R. E. Wall. *Mathematical Methods in Linguistics.* Kluwer, Dordrecht, 1990.

[Pechmann, 1989] T. Pechmann. Incremental speech production and referential overspecification. *Linguistics*, 27(1):89–110, 1989.

[Pereira and Shieber, 1987] F. C. N. Pereira and S. M. Shieber. *Prolog and Natural Language Analysis.* Center for the Study of Language and Information Stanford, 1987.

[Pereira and Warren, 1983] F. C. N. Pereira and D. Warren. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, Cambridge Massachusetts, 1983.

[Pollard and Sag, 1987] C. Pollard and I. A. Sag. *Information Based Syntax and Semantics, Volume 1.* Center for the Study of Language and Information Stanford, 1987.

[Pollard and Sag, to appear] C. Pollard and I. M. Sag. *Information Based Syntax and Semantics, Volume 2.* Center for the Study of Language and Information Stanford, to appear.

[Rayner, 1988] M. Rayner. Applying explanation-based generalization to natural language processing. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1988.

[Reithinger, 1991] N. Reithinger. Popel: A parallel and incremental natural language generation system. In C. L. Paris et al., editor, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 179–199. Kluwer, 1991.

[Rubinoff, 1988] R. Rubinoff. A cooperative model of strategy and tactics in generation. In *Paper presented at the Fourth International Workshop on Natural Language Generation*, Santa Catalina Island, 1988.

[Russell *et al.*, 1990] G. Russell, S. Warwick, and J. Carroll. Asymmetry in parsing and generating with unification grammars: Case studies from ELU. In *28th Annual Meeting of the Association for Computational Linguistics*, University of Pittsburgh, 1990.

[Samuelsson and Rayner, 1991] C. Samuelsson and M. Rayner. Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *IJCAI-91*, pages 609–615, Sydney, Australia, 1991.

[Samuelsson, 1994] C. Samuelsson. *Fast Natural-Language Parsing Using Explanation-Based Learning.* PhD thesis, Swedish Institute of Computer Science, Kista, Sweden, 1994.

[Schabes, 1990] Y. Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, USA, 1990.

[Schwarz, 1992] M. Schwarz. *Einführung in die Kognitive Linguistik.* UTB 1636, Franke Verlag Tübingen, 1992.

[Shieber and Schabes, 1990] S. M. Shieber and Y. Schabes. Synchronous tree–adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, Helsinki, 1990.

[Shieber *et al.*, 1983] S. M. Shieber, H. Uszkoreit, F. C. N. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In B. J. Grosz and M. E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*. SRI report, 1983.

[Shieber *et al.*, 1989] S. M. Shieber, F. C. N. Pereira, G. van Noord, and R. C. Moore. A semantic-head-driven generation algorithm for unification based formalisms. In *27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, 1989.

[Shieber *et al.*, 1990] S. M. Shieber, F. C. N. Pereira, G. van Noord, and R. C. Moore. Semantic-head-driven generation. *Computational Linguistics*, 16(1), 1990.

[Shieber, 1985] S. M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *23th Annual Meeting of the Association for Computational Linguistics*, Chicago, 1985.

[Shieber, 1988] S. M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

[Shieber, 1989] S. M. Shieber. *Parsing and Type Inference for Natural and Computer Languages*. PhD thesis, Standford University, SRI International Technical note 460, 1989.

[Smolka, 1988] G. Smolka. A feature logic with subsorts. Technical report, IBM Deutschland GmbH, Germany, 1988. Lilog-Report 33.

[Smolka, 1992] G. Smolka. Feature constraint logics for unification grammars. *The Journal of Logic Programming*, 12:51–87, 1992.

[Steele, 1990] G. L. Steele. *Common LISP: The Language (Second Edition)*. Digital Press, Burlington, MA, 1990.

[Strzalkowski, 1989] T. Strzalkowski. Automated inversion of a unification parser into a unification generator. Technical report, Courant Institute of Mathematical Sciences, New York University, 1989. No 465.

[Uszkoreit *et al.*, 1994] H. Uszkoreit, R. Backofen, S. Busemann, A.K. Diagne, E. Hinkelman, W. Kasper, B. Kiefer, U. Krieger, K. Netter, G. Neumann, S. Oepen, and S. Spackman. DISCO — an hpsg-based nlp system and its application for appointment scheduling. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, Kyoto, Japan, 1994.

[Uszkoreit, 1986a] H. Uszkoreit. Categorial unification grammar. In *Proceedings of the 11th International Conference on Computational Linguistics (COLING)*, Bonn, 1986.

[Uszkoreit, 1986b] H. Uszkoreit. Syntaktische und semantische generalisierungen im strukturierten lexikon. In C.-R. Rollinger and W. Horn, editors, *GWAI-86 und 2. Österreichische Artificial-Intelligence-Tagung, Ottenstein/Niederösterreich, Sept. 1986*, pages 87–100. Springer, Berlin, Heidelberg, 1986.

[Uszkoreit, 1991] H. Uszkoreit. Strategies for adding control information to declarative grammars. In *29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, 1991.

[VanNoord, 1993] G. J. M. VanNoord. *Reversibility in Natural Language Processing.* PhD thesis, University of Utrecht, The Netherlands, 1993.

[Vaughan and McDonald, 1986] M. M. Vaughan and D. D. McDonald. A model of revision in natural language generation. In *24th Annual Meeting of the Association for Computational Linguistics*, pages 90–96, 1986.

[Wahlster and Kobsa, 1986] W. Wahlster and A. Kobsa. Dialog-based user models. Technical Report 3, Project XTRA, Department of Computer Science, University Saarbrücken, Germany, 1986.

[Wahlster *et al.*, 1991] W. Wahlster, E. André, W. Graf, and T. Rist. Designing illustrated texts: How language production is influenced by graphics generation. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 8–14, Berlin, 1991.

[Wahlster, 1986] W. Wahlster. The role of natural language in knowledge-based systems. In H. Winter, editor, *Artificial Intelligence and Man-Machine Systems*, pages 62–83. 1986.

[Wahlster, 1991] W. Wahlster. User and discourse models for multimodal communciation. In *Intelligent user interfaces*, chapter 3, pages 45–67. ACM Press, 1991.

[Wedekind, 1988] J. Wedekind. Generation as structure driven derivation. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

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

[Wilks, 1991] Y. Wilks. Where i coming from: The reversibility of analysis and generation in natural language processing. Computing Research Laboratory,New Mexico State University, 1991.

[Winston and Horn, 1989] P. H. Winston and B. K. P. Horn. *LISP: Third Edition.* Addison-Wesley, Reading, MA, 1989.

[Wirén and Rönnquist, 1993] Mats Wirén and Ralph Rönnquist. Fully Incremental Parsing. In *Proc. Third International Workshop on Parsing Technologies*, Tilburg, the Netherlands and Durbuy, Belgium, 1993.

[Wirén, 1992] Mats Wirén. Studies in Incremental Natural-Language Analysis. Linköping Studies in Science and Technology, Dissertation 292, Department of Computer and Information Science, Linköping University, Linköping, Sweden, 1992.

[Woods, 1986] W. A. Woods. Transition network grammars for natural language analysis. In B. J. Grosz, K. Sparck Jones, and B. L. Webber, editors, *Natural Language Processing*, pages 71–87. Kaufmann, Los Altos, CA, 1986.

[Zeevat *et al.*, 1987] H. Zeevat, E. Klein, and J. Calder. Unification categorial grammar. In Nicholas Haddock, Ewan Klein, and Glyn Morrill, editors, *Categorial Grammar, Unification Grammar and Parsing.* Centre for Cognitive Science, University of Edinburgh, 1987. Volume 1 of Working Papers in Cognitive Science.