Interleaving Natural Language Parsing and Generation Through Uniform Processing

Günter Neumann*

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany
neumann@dfki.uni-sb.de

Abstract

We present a new model of natural language processing in which natural language parsing and generation are strongly interleaved tasks. Interleaving of parsing and generation is important if we assume that natural language understanding and production are not only performed in isolation but also work together to obtain subsentential interactions in text revision or dialog systems.

The core of the model is a new uniform agenda-driven tabular algorithm, called $\mathcal{UTA}$. Although uniformly defined, $\mathcal{UTA}$ is able to configure itself dynamically for either parsing or generation, because it is fully driven by the structure of the actual input—a string for parsing and a semantic expression for generation.

Efficient interleaving of parsing and generation is obtained through *item sharing* between parsing and generation. This novel processing strategy facilitates the automatic exchange of items (i.e., partial results) computed in one direction to the other direction as well.

The advantage of $\mathcal{UTA}$ in combination with the item sharing method is that we are able to extend the use of memorization techniques to the case of an interleaved approach. In order to demonstrate $\mathcal{UTA}$'s utility for developing high-level performance methods, we present a new algorithm for *incremental self-monitoring* during natural language production.

*The paper has been acceppted for publication in the Journal *Artificial Intelligence*. The research underlying this paper was supported by research grants from the German Science Foundation in its Special Collaborative Research Program on Artificial Intelligence and Knowledge Based Systems SFB 314 (project BILD) and from the German Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMB+F) to the DFKI projects DISCO (FKZ ITW 9002), PARADICE (FKZ ITW 9403) and PARADIME (FKZ ITW 9704). I would like to thank the following people for the fruitful discussions: Karin Harbusch, Walter Kasper, Martin Kay, Klaus Netter, Gertjan van Noord, Vijay Shankar, Hans Uszkoreit, Harald Trost, Wolfgang Wahlster, Mats Wirén.

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 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 (e.g., [Frazier, 1982; Kempen and Hoenkamp, 1987; Jacobs, 1988; Shieber, 1988; Appelt, 1987; Alshawi and Crouch, 1992; VanNoord, 1993; Ristad, 1993]). Recent developments in constraint-based grammar theories—due to their declarative and formal status—demonstrate that grammar reversibility is computationally feasible.

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 [Pereira and Warren, 1983], [Shieber, 1988]. Here grammatical processing is performed by means of the same underlying deduction mechanism, which can be parameterised for the specific tasks in hand.

Interleaving Parsing and Generation Natural language processing based on a uniform deduction process has a formal elegance and leads to more compact systems. There is one further important 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 of parsing and generation is important if we assume that understanding and production of natural language are not only performed in isolation but can also work together to obtain subsentential interactions in text revision or dialog systems.

If we distinguish two principle ways of interleaving, namely where generation is used in support of parsing, and where parsing is used in support of generation, then interleaved parsing and generation means:

1. the use of one mode of operation for monitoring and controlling the other, and
2. the use of structures resulting from one direction directly in the other.

For example, during parsing of an utterance, generation can already take place for the just parsed parts, by taking into account the parsing results at a very early stage of processing. Wirén and Rönnquist [1993] have argued that such a combined view on parsing and generation—in particular following a uniform approach— are worthwhile for exploring highly interactive text-processing facilities such as structure-editing operations, propagation of minimal grammatical changes, or on-line translations, in which the target-language text is generated parallel to the source-language text [Somers *et al.*, 1990]. Self-control of the parsing process through interleaved generation is also important for handling under-specified or ill-formed input where generation is used to "guess" the missing parts or to perform some sort of repair work (e.g., to "guess" what the ill-formed utterance probably means). Clearly, additional knowledge-based mechanisms are needed for the realization of its full functionality, so that interleaved parsing and generation is only one step in that direction —but it is, however a substantial one.

During natural language production interleaved parsing is important to obtain hearer-adaptable production of utterances. Wahlster [1991] has expressed this under the term *anticipation feedback loop* AFL. The basic idea of the AFL model is the use of the system's natural language understanding component to anticipate the users' preferred interpretation of an utterance which the system plans to realize. In psycholinguistic research a similar strategy is known under the term *self-monitoring.* Here there is no denying that people are careful about what they say and how they say it [Berg, 1986]. The basic task of monitoring is to gain information about processing which is not necessarily obvious, i.e., a device is called which can make this information available to the speaker or the hearer. It has often been argued in cognitive psychology [Levelt, 1989] that it is highly desirable to find a mechanism that is an integral and independently motivated part of the whole system and that performs the monitoring function by its own nature. Kempen has noted that "...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. 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." (cf. [Kempen, 1989], page 15).

Such strategies would be very useful in practical systems which have to perform some sort of *ambiguity checks*, e.g., controlled language checking [Carnegie Group, 1994; Adriaens and Macken, 1995], text revision [Cline and Nutter, 1994], or in systems which have to produce brief speech in time-pressured caregivers [McKeown *et al.*, 1997]. In systems of those kinds, integrated parsing can be used to monitor the generation process and to cause some sort of revision *to reduce the risk of misunderstandings.* For instance, in the case of controlled language checking, interleaved parsing and generation can be used to find out whether an utterance of a current text lies outside the controlled language. If so, the generation process could re-use partial results already computed through parsing as well as the detected ambiguity sources in order to compute "proper" paraphrases. The advantage of using an interleaved approach is that the paraphrasing process only needs to re-configure already computed structures and that generation of irrelevant paraphrases can be avoided.

Interleaved parsing and generation also seems promising for question-answering systems where understanding and answering of questions is performed simultaneously [Robertson, 1994] and for bidirectional dialogue systems [Levine, 1992]. Here an interleaved approach can be used to efficiently model clarification dialogues like "Do you mean X or Y?" or, more generally, to explore new models of effective communication which are based on methods of incremental adaptation to a common language use between the interlocutors.

In nearly all of the above cited approaches, parsing and generation are assumed to work together on a very fine-grained level. In fact, if we can realize interleaving of parsing and generation in such an incremental way, the whole natural language system would achieve an important degree of self-control. It is our conviction that system immanent self-control is an important pre-requisite for achieving truly flexible and adaptable natural language systems—the core motivation of our uniform framework.

Modelling such high-level performance methods on the basis of non-uniform approaches is problematic—if not impossible. For example, if two different grammars and algorithms are in use, additional translation operations are necessary for parsing and generation to exchange partial results. Since this is a complex process in itself, not only maintaining two specific grammars but also two different algorithms, will be a handicap for an interleaved approach.

Unfortunately, the currently proposed uniform architectures are too inflexible and inefficient; so it seems unclear how an efficient task-oriented uniform model could be achieved. An obvious problem is that different input structures are involved in each direction—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.

The contribution of this work In this paper we present $\mathcal{UTA}$, a novel uniform algorithm for parsing and generation of constraint-based grammars that overcomes these problems. The most interesting properties of $\mathcal{UTA}$ are:

1. a uniform data-driven processing strategy,
2. item sharing between parsing and generation, and
3. a co-routine relationship between parsing and generation.

The first property means that parsing and generation are both realized by the single program $\mathcal{UTA}$, but that it is able to configure itself dynamically for either parsing or generation. The only *essential parameter* for $\mathcal{UTA}$ to adapt itself efficiently to either the parsing or generation task is the feature that carries the input, hence we call it the *essential feature* (Ef). This information suffices to define a data-driven selection function (Ef determines the selection of the next right hand side element of a rule), and a uniform chart mechanism (partial results are ordered according to the value of their Ef).

Secondly, $\mathcal{UTA}$ extends the traditional usage of a chart by allowing for *shared items* between parsing and generation: Partial results computed in one direction are automatically made available for the other direction as well. Thus, if parsing and generation work in tandem to solve a specific problem, they are capable of exchanging the result of partial computations, which reduces the amount of unnecessary computations in those cases. In other words, $\mathcal{UTA}$ extends the usage of a chart to the extent that parsing and generation are strongly interleaved.

Interleaving of parsing and generation is realized with a *co-routine processing* regime between both directions using a flexible agenda mechanism. Here, parsing and generation are considered as specific instances of $\mathcal{UTA}$. We call the different instances *parser* and *generator* (but note that both are realized by the same algorithm). The only differences are 1.) different values for the essential feature Ef, and 2.) each one has its own individual agenda. The agenda control—which is the same for both—is able to co-routine between both directions in a *fine-grained incremental* manner. For example, during parsing generation is called for a just analysed partial string. The result of the generator may then influence parsing of the following partial strings. Obviously, this complex processing strategy benefits directly from the item sharing mechanism introduced above. As another example, we show in detail how integrated parsing is used during generation for *incremental self-monitoring* of the generation process. It will be demonstrated that such a complex process can be realized quite easily and efficiently using $\mathcal{UTA}$'s novel properties.

$\mathcal{UTA}$ and the incremental monitoring strategy have been fully implemented in CommonLisp and CLOS and tested with constraint-based lexicalized grammars for Dutch and German. It uses the powerful constraint-solver $\mathcal{UD}i\mathcal{N}e$, which is capable of dealing with distributed disjunctions over arbitrary structures, negative co-references, and full negation [Uszkoreit *et al.*, 1994].

Overview of the following sections In the next section we introduce the formal and linguistic background on which our approach is based. In particular we discuss abstractly the notions of reversible grammars and uniform algorithms, and introduce *constraint logic programming* (CLP) as an appropriate means for establishing the computational basis of uniform processing. In section 3 we describe the new uniform tabular algorithm in detail, and discuss some of its properties briefly in section 4. Section 5 then presents the item sharing approach between parsing and generation. On the basis of $\mathcal{UTA}$ and item sharing, section 6 demonstrates how interleaved parsing and generation is used to realize an incremental self-monitoring strategy. In section 7 we compare the new approach to related work and outline future extensions.

2 Formal and Linguistic Background

2.1 A relational view on language

It is widely accepted to consider linguistic objects (i.e., words and phrases) as utterance-meaning associations [Pollard and Sag, 1994]. Thus viewed, a grammar is a formal statement of the relation between utterances of a natural language and representations of their meanings in some logical or other artifical language, where such representations are usually called *logical forms* [Shieber, 1993].

Adopting the simplified assumption that utterances are represented as strings of words, the relationship can be defined more formally 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 logical forms. 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 logical forms. More precisely, in the case of parsing P computes $\{lf_i | \langle s, lf_i \rangle \in R, i = 1 \ldots n\}$, and in the case of generation $\{s_i | \langle s_i, lf \rangle \in R, i = 1 \ldots m\}$. Thus P is just a constructive realization of R, no matter whether P constructs R during parsing alone or during generation. We call P a *reversible* program and R a *P-reversible* relation, in order to emphasize that P can construct R from both directions.

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 also enumerate R from the set LF. Furthermore, we also assume that S has an infinite cardinality, so that R has to be defined by some finite recursive device, i.e., a grammar. If the same grammar is used for defining both sets of R, we call this grammar a *reversible grammar*.

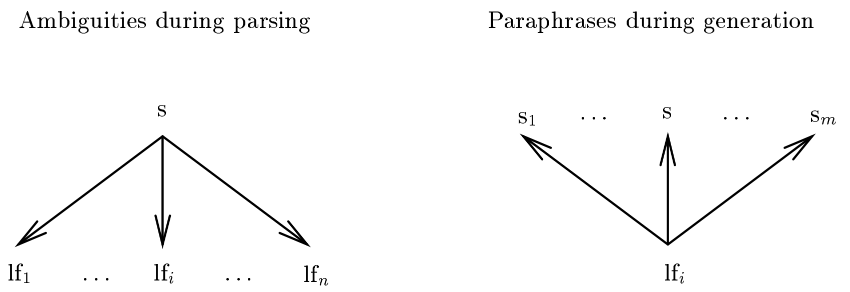

Figure 1: The relationship between ambiguities and paraphrases.

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 logical form 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 $\mathrm{card}(R(s))$ is defined as the *degree of ambiguity* of s and $\mathrm{card}(R(lf))$ as the *degree of paraphrasing* 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 1). 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; 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 sentences, cannot be modelled properly (see section 6).

Thus viewed, understanding and production 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.

2.2 Constraint-logic programming

Since the last decade a family of linguistic theories known under the term *constraint-based grammar theories* has begun to play an important role within the field of natural language processing, e.g., LFG[Bresnan, 1982], HPSG[Pollard and Sag, 1994].

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

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 specification

$sign(X_0) \leftarrow$
$\quad sign(X_1),$
$\quad sign(X_2),$
$\quad X_0\ syn\ cat\ \doteq\ s,$

$$\begin{aligned} X_1 \ syn\ cat &\doteq np, \\ X_2 \ syn\ cat &\doteq vp, \\ X_1 \ syn\ agr &\doteq X_2 \ syn\ agr \end{aligned}$$

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 them 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.

A general characterisation of CLP is provided in [Höhfeld and Smolka, 1988]. 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.

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 ϕ 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 $q \leftarrow p$. If the head and body of a clause are empty, we call it an *empty clause*. 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, in particular 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 generalisation of the SLD-resolution method known from standard logic programming (cf. [Lloyd, 1987]) to definite clauses in $\mathcal{R}(\mathcal{L})$.

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])

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

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 result of unifying ϕ and ψ (which we also write as $\text{UNIFY}(\phi, \psi)$).[1]

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.

Constraint language The constraint language we use 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 work it suffices to use only a small subset of Smolka's constructions, namely feature equation and conjunction.[2]

We will not give a formal definition of the constraint language here since this has already been done (see [Smolka, 1992; VanNoord, 1993]). Instead we make direct use of the "Prolog-flavoured" matrix representation introduced by Van Noord as a readable notation of $\mathcal{L}$-constraints.[3] For example, the following constraints on the variable X_0

$X_0\ f_1\ f_3 \doteq \mathrm{c}$,

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

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

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

If variables occur only once in a matrix they are omitted. Furthermore, empty feature structures will not been shown explicitly. The feature structure encoding of the following list

[1] Note that we make direct use of the so called *optimised goal reduction rule* proven by [Höhfeld and Smolka, 1988] for the general case.

[2] Although we only use simple constructions in order to highlight the new results in a clean but simple way, the generalisation of Höhfeld and Smolka's scheme guarantees that the results of this paper also carry over to more complex constraint languages. Note further that the same subset has also been used by [VanNoord, 1993] (for the same reasons).

[3] The only important thing to note here is that the constraints are based on disjointed sets of variables, constants, and features, as well as descriptor equations, where a descriptor is a (possibly empty) sequence of features starting with a variable or a constant. The semantics of $\mathcal{L}$-constraints is defined with respect to the domain of feature graphs.

$$(2)\quad \begin{bmatrix} \texttt{first}\ a \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ b \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ c \\ \texttt{rest}\ \ end \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

will be made more readable by use of angled brackets, e.g. $\langle a\ b\ c\rangle$. An empty list will then be written as $\langle\ \rangle$.

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 $\langle First|Rest\rangle$ (e.g., $\langle a,\ b,\ c\rangle$ can also be written as $\langle a|\langle b,\ c\rangle\rangle$). Using this notation the difference list of the feature structure

$$(3)\quad \begin{bmatrix} \texttt{dl}\ \begin{bmatrix} \texttt{first}\ a \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ b \\ \texttt{rest}\ \begin{bmatrix} \texttt{first}\ c \\ \texttt{rest}\ \ X \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \texttt{el}\ X \end{bmatrix}$$

will be written as $\langle a\ b\ c|X\rangle$-X , and the empty difference list as X-X .

2.3 Specification of grammatical knowledge in $\mathcal{R}(\mathcal{L})$

A grammar G is specified as a definite clause specification where the literals of each definite clause are unary relational atoms.[4] The general form of a *grammar rule* is as follows:

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

This rule can also be represented as

$$p(fs_0)\leftarrow q_1(fs_1)\ldots q_n(fs_n)$$

where fs_i is the feature structure representation of the corresponding variable x_i.

Lexical entries are represented as unit clauses, and grammar rules as non-unit clauses (defining non empty productions) as well as unit clauses (defining empty productions). Lexical entries and empty productions are distinguished using the boolean feature LEX.

Relational atoms are assumed to denote possible constituents of a grammar, either specifically (using a specific symbol for each possible constituent, like *np, vp, pp*) or schematically by only using one symbol, e.g., *sign*. For example, the rule

[4]Considering only unary atoms is not a general restriction since by means of reification we can also express an n-ary atom $r(\vec{X})$ 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 $r(\vec{X})$ would be represented as follows: $s(Y), Y\ rel\ \doteq\ r, Y arg_i\ \doteq\ X_i$.

$$sign(fs_1) \leftarrow sign(fs_2),\ sign(fs_3)$$

expresses that a phrase is built up of two phrases, no matter what they are (as long as we do not consider the feature structure). 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*, as they allow the specification of general combinatory rules, which are independent from individual words.

2.4 Parsing and generation under a CLP view

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 entail at least 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*, Ef for short. For parsing we will assume that Ef is the path $\langle$PHON DL$\rangle$ and for generation it is $\langle$SEM$\rangle$.

A parsing goal can then be defined as a goal of which the essential feature is $\langle$PHON DL$\rangle$ and whose value is bound to the string in question. For example, the parsing problem for the string "heute erzählt Peter Lügen" (*today, Peter is telling lies*) 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 $\langle$SEM$\rangle$ and whose value is bound to the semantic expression in question. For example, the generation problem for the logical form "heute(erzählen(Peter,Lügen))" (*today(to_tell(Peter,lie))*) would be

$$sign(\begin{bmatrix}\texttt{sem} & \begin{bmatrix}\texttt{mod} & heute \\ \texttt{arg} & \begin{bmatrix}\texttt{pred} & erzählen \\ \texttt{arg1} & \begin{bmatrix}\texttt{pred} & Peter\end{bmatrix} \\ \texttt{arg2} & \begin{bmatrix}\texttt{pred} & Lügen\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix})$$

In both cases further constraints may be added to restrict the possible feature structures of found results, perhaps to be of a specific category, or that the subcategorization list should be empty. It would also be possible to specify all the syntactic information in the case of generation, to perform some grammar checking. However, what the least we require is that the value of the essential feature is instantiated for parsing and generation.

Restricted parsing problem So far, we have only required that the essential feature should be instantiated. More precisely, we want our algorithm to enumerate all possible feature structures that have a *compatible* value for the 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.

Van Noord [1993] 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. More formally, the p-parsing problem consists of a grammar G and a goal q such that $\leftarrow q(X), \phi$. An answer to a p-parsing problem is a solved constraint ψ such that

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

(where $[\![(\phi/Xp)]\!]^{\mathcal{I}}$ is the subgraph found under the path p). In our terminology the path p corresponds to the essential feature Ef. Thus we also use the term *Ef-proof problem* to indicate that parsing and generation are proofs of goals in which the essential feature is instantiated.

3 $\mathcal{UTA}$–A new Uniform Tabular Algorithm

We will now introduce $\mathcal{UTA}$—a new uniform tabular algorithm for parsing and generation with constraint-based grammars. $\mathcal{UTA}$'s basic use is for parsing and generation of grammatical structures. Apart from this more traditional use, $\mathcal{UTA}$'s new potential emerges when it is used in such high-level processing strategies which are based on a tight interaction or interleaving of parsing and generation. $\mathcal{UTA}$ will be described along the following lines:[5]

- data-driven selection function
- uniform indexing mechanism
- agenda-based control
- item sharing between parsing and generation

[5] The parsing and generation examples of Appendix A will be used for the illustration of the main notions introduced throughout the next sections.

3.1 Data-driven selection function

The discussion of current approaches for parsing and generation can be summarised 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, [Shieber, 1988; Gerdemann, 1991] for example have used the leftmost selection strategy, while [Shieber *et al.*, 1990; Gerdemann, 1991] use the semantic-head first selection function for generation. The latter should be seen more precisely as a "preference-based" selection function, since if 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 easy to combine these different strategies used in parsing and generation, such that the selection function expresses a preference for goals with their essential features instantiated. If we abstract away from a concrete essential feature by assuming that Ef is a variable, then we can define this selection function more formally as follows:

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

In order to use the 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. Now, the grammar itself will be an important source of control, as it defines how complex structures are compositionally created. 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. Both of these can be true at the same time.

3.2 Uniform indexing mechanism

The purpose of the indexing mechanism employed by $\mathcal{UTA}$ is threefold:

1. avoidance of redundant recomputation by memoing just analysed clauses (i.e., parsed or generated),
2. splitting of derived clauses into equivalence classes so that necessary lookup operations are restricted to an identifiable subset,
3. use of the same mechanism for both parsing and generation.

The idea of memoing derived clauses as well as defining equivalence classes for restricting lookup of possible candidates is not a new one (cf. [Earley, 1970; Pereira and Warren, 1983; Kay, 1986]) although here primary emphasis was put on parsing. However, considering memorization under a strictly uniform and interleaved perspective as is followed in this paper, has not been described, to the best of my knowledge.[6]

For parsing, particular data structures have been developed to achieve efficient processing, 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 directly use these well-known approaches for generation, because the string is the output of a generator, not the input. For generation, once a phrase has been constructed, we want to be able to use it at various places.

We will 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 parameterised. Since the only parameter is a certain feature and its value, we have achieved *a maximal degree of uniformity for parsing and generation under a task-oriented view.*

The structure of items $\mathcal{UTA}$'s indexing mechanism is based on two data structures, viz *item* and *item set*. An item records the current state of a derived clause. We have to distinguish a clause whose body is not empty from one whose body is empty. The latter will be called *passive clause* and the former *active clause*. In the same sense we distinguish *passive items* from *active items*. An *active item* is of the form:

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

where $h \leftarrow b_0 \ldots b_n$ is an active clause, i $(0 \leq i \leq n)$ is the index of the selected element in the body of the active clause, and idx is the value of the essential feature of the selected element. The selected element is determined by applying the selection function SF to the active clause.[7]

The general structure of a *passive item* is of the form

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

where h is a passive clause, and idx the value of the essential feature of the head h. ϵ indicates that no selected element can be determined since the body is empty , and hence the selection function should not be applied.

[6] Martin Kay (p.c.) is currently also investigating uniform indexing mechanisms for parsing and generation, but not under an interleaved perspective.

[7] As long as no misunderstandings are possible, we will use the terms "selected element" and "index of selected element" in the same sense.

For the representation of the *start item* (i.e., from which processing of a parsing or generation query q starts) we specify the goal statement q as the negative literal of an $\mathcal{R}(\mathcal{L})$-atom that does not belong to the grammar or the lexicon. Thus the structure of the start item is as follows (because q is the only element of the body its index is 0):

$$\langle ans(fs_s) \leftarrow q(fs_s);\ 0;\ fs_s/Ef \rangle$$

Thus, the index is either the string or semantic input in question. Note that the constraints fs_s of q are shared between q and ans. Hence the structure of a *goal item* is as follows:

$$\langle ans(fs_g);\ \epsilon;\ fs_g/Ef \rangle$$

i.e., the goal item's clause is passive. Because of the Ef-proof problem it yields that $[\![(fs_s/Ef)]\!]^{\mathcal{I}} = [\![(fs_g/Ef)]\!]^{\mathcal{I}}$. Consequently, the start item and the goal items have the same index; e.g., in the parsing example in Appendix A (figure 3) the start item 1 (referring to an item's left index) and the goal items 16 and 22 are placed in the same item set whose index is sPmM_0.

The structure of item sets The basic idea is to split the generated items into equivalence classes and to connect these classes, so that each item can be directly restricted to those items that belong 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 use the index *idx* of an item as index for an item set. Note that *idx* equals the value of the essential feature of the selected element of the item's clause (abbreviated as VEF)). We require that for each item L in an item set I with index Idx, VEF(L,Ef) must be the same as Idx. 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:

$$\begin{aligned} \forall\ pl_i, pl_j \in PL : \text{VEF}(pl_i, Ef) &= \text{VEF}(pl_j, Ef) = Idx \text{ and} \\ \forall\ al_i, al_j \in AL : \text{VEF}(\text{SEL}(al_i), Ef) &= \text{VEF}(\text{SEL}(al_j), Ef) = Idx \end{aligned}$$

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 a passive clause, and the selected element in the case of an active clause (see figure 3 and 4 in Appendix A).

In this sense, an item set can be viewed as a kind of *meeting place* for active and passive items, where an active item looks for a passive item to resolve, 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.

It is important to note that the different item sets are implicitly structured according to the structure of the actual input. For example, if the phonological information is represented as a list, then the item sets are also ordered in a list-like manner. If, on the other hand, a tree-like semantic representation is used in the grammar, then the structure of the item sets also bears a tree-like structure. This is due to the fact that the value of the essential feature is used to define the indexes of the item sets.

3.3 Inference-rules

The control logic of $\mathcal{UTA}$ is a generalisation of the Earley deduction scheme as introduced by [Pereira and Warren, 1983] (see also [Pereira and Shieber, 1987]). It is similar to the one defined by [Shieber, 1988] with the notable distinction that we use a dynamic selection function (where Shieber only uses the left-to-right selection function for both parsing and generation), and that we use a fairly uniform indexing mechanism (where Shieber only uses indexing efficiently for parsing because his indexing scheme is explicitly based on string positions). Furthermore our approach is the first that makes use of *shared items* between parsing and generation (see section 5).

$\mathcal{UTA}$ operates on two sets of definite clauses, called the *grammar* and the *chart*. The grammar simply represents the grammar rules and lexical entries and remains fixed. They are kept in two different data bases (called *Rules* and *Lex* respectively) to enable efficient retrieval. The chart on the other hand, will be continually augmented with new derived clauses, i.e. lemmas. Whenever a new active lemma is added to one of the chart's item sets, one of its negative literals is selected by calling the selection function SF, i.e., a selected element is determined *on-line*.

Following [Pereira and Warren, 1983] we make use of the following inference rules: *prediction* and *completion*. *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, passive clauses will be used to reduce appropriate active clauses, where the scanning rule can be seen as a special active completion rule in the sense that it looks for passive clauses in the lexicon which it uses to reduce the active clause in question.

Using the uniform indexing technique the inference rules can be described as follows. Note that each time a new item Ni is deduced by an inference rule, two additional things will happen. Firstly, a new empty item set I with the index of Ni will be created if it does not already exist. This means, that item sets are created *on-line*. Secondly, Ni is not added to I, but to the agenda *Agenda* according to a determined priority using the function PRIO. This means that a newly created item set I remains empty until the agenda mechanism has chosen an item for I.

Prediction Let $\langle h \leftarrow b_0 \dots b_n;\ i;\ idx \rangle$ be an active item Ai. Then

$prediction(Ai)$ is:

For every rule $R \in Rules$:
  if $\Phi = \text{UNIFY}(\text{ABSTRACT}(\text{SEL}(Ai)), \text{HEAD}(R))$ **and** $\Phi \neq fail$ **then**
    with new lemma $Nl = \Phi[R]$ **do**
    if $\text{BODY}(Nl) \neq \epsilon$ **then**
      make new active item with $Selem = \text{SF}(Nl, Ef)$:
        $Ni = \langle Nl;\ Selem;\ Selem/Ef \rangle$
    else
      make new passive item:
        $Ni = \langle Nl;\ \epsilon;\ \text{HEAD}(Nl)/Ef \rangle$
    fi;
    create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
    ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
    od fi.

Here, the selected element of Ai and the rule's head element (the left-hand-side element) are unified, and only if unification was successful will a new item be created. Thus, prediction deduces a new item on the basis of an instantiated rule, e.g, using the selected element of the start item 1 (see figure 3, Appendix A), the new tasks 1 and 2 are created and added to the agenda.[8] No item set will be created because I_{sPmM0} already exists.

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. In order to avoid such problems, prediction should be performed with an abstraction of the selected element's constraints (which is determined by the function ABSTRACT).[9]

Scanning Let $\langle h \leftarrow b_0 \ldots b_n;\ i;\ idx \rangle$ be an active item Ai. Then

scanning(Ai) is:

For every lexical entry $L \in Lex$:
  if $\Phi = \text{UNIFY}(\text{SEL}(Ai), \text{HEAD}(L))$ **and** $\Phi \neq fail$ **then**
    with reduced lemma $Rl = \Phi[Ai - \text{SEL}(Ai)]$ **do**
    if $\text{BODY}(Rl) \neq \epsilon$ **then**
      make new active item with $Selem = \text{SF}(Rl, Ef)$:
        $Ni = \langle Rl;\ Selem;\ Selem/Ef \rangle$

[8] Because in Appendix A we only follow successful derivations, not all possibly predictable items are considered.

[9] We follow an approach similar to the one described in [Johnson and Dörre, 1995], by generalising the value of only a small predefined set of constraints, namely those which are known to cause termination problems. The advantage of our approach is that we are able to perform prediction with as many constraints as possible from the selected element. In parsing literature abstraction has been introduced under the term "restriction". For more and detailed information on the definition and use of an abstraction/restriction function during parsing see e.g., [Shieber, 1985], [Haas, 1989], and [Samuelsson, 1994].

else
make new passive item:
$Ni = \langle Rl;\ \epsilon;\ \text{HEAD}(Rl)/Ef)\rangle$
fi;
create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
od fi.

Thus, if a lexical entry can be unified with the selected element of the active item Ai, then a new clause is constructed by deleting the unified element from the body of Ai's clause. Following [Pereira and Warren, 1983], we call this deletion operation *reduction*. As an example of scanning, consider item 2 (see figure 3, Appendix A), which is used to scan the input word "sieht" (*to see*). The resulting reduced task 3 is added to the agenda (creating the initially empty item set I_{PmM1}).

The same will be performed for the two remaining completion rules, passive completion and active completion. Thus, by successive application of the completion rules an active item can be transformed into a passive item for which reduction will no longer be possible.

Passive-completion Let $\langle h;\ \epsilon;\ idx\rangle$ be a passive item Pi. Then

p-completion(Pi) is:
For every active item $Ai \in I_{idx}$:
if $\Phi = \text{UNIFY}(\text{SEL}(Ai),\ h)$ **and** $\Phi \neq fail$ **then**
with reduced lemma $Rl = \Phi[Ai - \text{SEL}(Ai)]$ **do**
if $\text{BODY}(Rl) \neq \epsilon$ **then**
make new active item with $Selem = \text{SF}(Rl, Ef)$:
$Ni = \langle Rl;\ Selem;\ Selem/Ef\rangle$
else
make new passive item:
$Ni = \langle Rl;\ \epsilon;\ \text{HEAD}(Rl)/Ef\rangle$
fi;
create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
od fi.

Passive completion is only applied to active items which are in the same item set as the passive item. For example, if passive item 13 placed in item set I_{mM2} is selected (see figure 3, Appendix A), then it will use active item 8, found in the same item set for completion. The resulting task 16 is added to the agenda and, at a later time of processing, inserted as passive item 14 into the item set I_{PmM1}.

Finally, the definition of active completion is

Active-completion Let $\langle h \leftarrow b_0 \ldots b_n;\ i;\ idx \rangle$ be an active item Ai. Then

a-completion(Ai) is:

For every passive item $Pi \in I_{idx}$:
if Φ = UNIFY(SEL(Ai),HEAD(CLAUSE(Pi))) **and** $\Phi \neq fail$ **then**
with reduced lemma $Rl = \Phi[Ai - \text{SEL}(Ai)]$ **do**
if BODY(Rl) $\neq \epsilon$ **then**
make new active item with $Selem = \text{SF}(Rl, Ef)$:
$Ni = \langle Rl;\ Selem;\ Selem/Ef \rangle$
else
make new passive item:
$Ni = \langle Rl;\ \epsilon;\ \text{HEAD}(Rl)/Ef \rangle$
fi;
create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
od fi.

3.4 Agenda-based control

The inference rules will be embedded in an agenda-based control regime along the lines 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 a priority function PRIO. If we name the agenda mechanism *process* and the query to prove G then

process(G, Ef) is:

make start item Si using G;
ADD-TASK-TO-AGENDA(Si,PRIO(Si),$Agenda$);
while NOT(EMPTY-AGENDA-P($Agenda$)) **do**
let current task Ct = GET-HIGHEST-PRIO-TASK($Agenda$);
if ADD-ITEM(Ct) **then do**
if Ct is a goal item **then**
add Ct to result list Res **fi**
APPLY-TASK(Ct) **od od**;
if $Res = \emptyset$ **then** return rejection
else return Res
fi.

where *add-item*$(Item)$ is:

if $\neg \exists I \in I_{\text{INDEX}(Item)}$: I subsumes $Item$
then add $Item$ to $I_{\text{INDEX}(Item)}$
fi.

and *apply-task*$(Item)$ is:

if *Item* is passive **then**
P-COMPLETION(*Item*) **else**
A-COMPLETION(*Item*) **else do**
PREDICTION(*Item*);
SCANNING(*Item*)
od.

The way the agenda sorts new tasks depends on the priority assigned to each newly created item. Hence the priority function determines the search strategy. In our current system we are using generic search strategies which refer to a task counter. Direct use of the value of this counter realizes a depth-first strategy, since each new task is added to the front of the queue (see the examples in Appendix A). Using its negative value instead would realize a breadth-first strategy, because processing of new items is delayed until older tasks are processed. In addition to a depth-first and breadth-first task-selection function, we have also defined a version of 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 (see section 4 for information).

The function ADD-ITEM performs insertion of an item into the chart. The appropriate item set is selected using the index of the new item. Before the new item is added to that item set, it is checked whether there is already an element in that item set which subsumes the new item. This test is known as the *blocking* test [Pereira and Warren, 1983]. Although, we currently use the expensive subsumption operation for this test, our uniform indexing mechanism makes it possible to apply subsumption to a restricted subset of all possible items already in the chart. Additionally, the agenda mechanism selects only those items which are currently considered as relevant for one proof. This is important not only if we follow a best-first search strategy, but in particular when we are going to interleave parsing and generation.

The inference rules are called inside the function APPLY-TASK. If the current item is passive, then passive completion is applied. Otherwise active completion is called. The reason why we only consider prediction and scanning if active completion returns false (i.e., creates no new items), is that if active completion is successful this means that there already exists a derived phrase for the selected element of the current active item (made for the same substring or partial semantics), and hence prediction and scanning would be redundant.[10]

[10] It is 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 being is that in general, constraint-based grammars are under-specified in this respect. 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.

3.5 Parsing and generation with $\mathcal{UTA}$

In order to run $\mathcal{UTA}$ for parsing or generation we only need to specify a query q which contains the input and the value of the essential feature Ef, i.e., the path to the input string or the semantics. For parsing we choose the feature $\langle$PHON DL$\rangle$ and for generation we choose the path $\langle$SEM$\rangle$.

Then $parse(q)$ is:
$\textsc{process}(q, \langle \textsc{phon dl} \rangle)$.

and $generate(q)$ is:
$\textsc{process}(q, \langle \textsc{sem} \rangle)$.

For the examples given in section 2.4 the call of PROCESS for parsing looks like

$$\textsc{process}(sign(\begin{bmatrix}\texttt{phon} & \langle heute, erz\ddot{a}hlt, Peter, L\ddot{u}gen\rangle\text{-}\langle\ \rangle\end{bmatrix}), \\ \langle heute, erz\ddot{a}hlt, Peter, \text{Lügen}\rangle)$$

and for generation it looks like

$$\textsc{process}(sign(\begin{bmatrix}\texttt{sem} & \begin{bmatrix}\texttt{mod} & heute \\ \texttt{arg} & \begin{bmatrix}\texttt{pred} & erz\ddot{a}hlen \\ \texttt{arg1} & \begin{bmatrix}\texttt{pred} & Peter\end{bmatrix} \\ \texttt{arg2} & \begin{bmatrix}\texttt{pred} & L\ddot{u}gen\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}), \begin{bmatrix}\texttt{mod} & heute \\ \texttt{arg} & \begin{bmatrix}\texttt{pred} & erz\ddot{a}hlen \\ \texttt{arg1} & \begin{bmatrix}\texttt{pred} & Peter\end{bmatrix} \\ \texttt{arg2} & \begin{bmatrix}\texttt{pred} & L\ddot{u}gen\end{bmatrix}\end{bmatrix}\end{bmatrix})$$

If we assume a grammar capable of processing these examples, where strings are represented through concatenation and semantic expressions through predicate/argument trees, then the indices of the created item sets are for parsing (specified in the order in which they are created during processing):

$\langle heute, erz\ddot{a}hlt, Peter, \text{Lügen}\rangle$; $\langle erz\ddot{a}hlt, Peter, \text{Lügen}\rangle$; $\langle Peter, \text{Lügen}\rangle$; $\langle \text{Lügen}\rangle$; $\langle\ \rangle$

and for generation:

$$\begin{bmatrix}\texttt{mod} & heute \\ \texttt{arg} & \begin{bmatrix}\texttt{pred} & erz\ddot{a}hlen \\ \texttt{arg1} & \begin{bmatrix}\texttt{pred} & Peter\end{bmatrix} \\ \texttt{arg2} & \begin{bmatrix}\texttt{pred} & L\ddot{u}gen\end{bmatrix}\end{bmatrix}\end{bmatrix}; \begin{bmatrix}\texttt{pred} & erz\ddot{a}hlen \\ \texttt{arg1} & \begin{bmatrix}\texttt{pred} & Peter\end{bmatrix} \\ \texttt{arg2} & \begin{bmatrix}\texttt{pred} & L\ddot{u}gen\end{bmatrix}\end{bmatrix}; \begin{bmatrix}\texttt{pred} & Peter\end{bmatrix}; \begin{bmatrix}\texttt{pred} & L\ddot{u}gen\end{bmatrix}$$

In Appendix A complete parsing and generation examples can be found.

4 Intermezzo: Some Properties of $\mathcal{UTA}$

$\mathcal{UTA}$ is a straight-forward extension of the optimized general SLD-resolution rule whose correctness is proven in [Höhfeld and Smolka, 1988]. It also inherits this property (see [Neumann, 1994b] for more details).

Since $\mathcal{UTA}$ prefers in each deduction step those clauses whose selected element's Ef is instantiated it has a very strong goal-directed as well as data-oriented behaviour in particular for the case of generation. The only relevant parameter our algorithm has with respect to parsing and generation is the difference in input structures. Thus we are able to characterise parsing and generation in a fairly balanced way without the loss of efficiency 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. In [Neumann, 1994b] we also show how $\mathcal{UTA}$ is extended to handle empty heads, which are used to describe verb second constructions in Germanic languages such as Dutch and German.

$\mathcal{UTA}$ handles efficiently coherence and completeness as required by the Ef-proof problem (which means that only and all elements of the input are considered during a proof, see 2.4). The core idea is that only unit clauses which actually cover or consume parts of the input structure are considered during scanning and that the goal item is placed in the same item set as the start item (for more details see [Neumann, 1994b]).

In this paper we have made use of a rather simple form of semantic representation for the purpose of illustration. However, $\mathcal{UTA}$ will also work for more complex semantic forms as long as they are representable in a constraint-based formalism [Nerbonne, 1992].[11] The only major restrictions we have on a semantic formalism are the following: It must be compatible with a sign-based approach of grammar theory (see [Pollard and Sag, 1987; Pollard and Sag, 1994]). In the case that some lexical entries do not have any semantic information we require that these entries have a semantic value NIL defining the "null" semantics. This not only facilitates indexing of the lexicon but also prevents unification of "spurious" semantic information for such entries which could lead to erroneous results. Furthermore, we require that the start and end items are in the same item set, which implies that their semantic information must be equal. Thus we will not be able to handle underspecified semantic input, at least without modifications of $\mathcal{UTA}$'s indexing mechanism. However, if the semantic formalism were to come with a richer notion of comparison (e.g., analogy or similarity), we might be able to improve the indexing schema.

The uniform chart mechanism together with the agenda-based control support the implementation of methods that go beyond simple parsing and generation, as we will demonstrate in the next two sections. In particular the on-line creation of item sets supports incremental processing for both parsing and generation, and even for the

[11] The author has successfully used the Montague-style semantic representation described in [Pereira and Shieber, 1987], and the newly developed *minimal recursion semantics* (MRS) described in [Copestake *et al.*, 1996]; how MRS is used for generation can be found in [Kay, 1996; Neumann, 1997].

interleaved approach. In the same spirit, the agenda mechanism supports integration of more complex priority functions which take into account, for instance, the source of an item (e.g., whether it is a predicted, lexical or completed one), an item's index (e.g., sort items according to longest span), the structure of rules (e.g., prefer rules with shorter right-hand side), or information on an item's internal feature structure (see [Shieber, 1988; Barnett, 1994] for more information). Following Barnett we assume that priority functions are very efficient because they have to be evaluated automatically every time a new task has been created. However, functions which are also used to control the search space, but which are based on complex strategies (like the self-monitor described in section 6) and which will not run automatically at each step during parsing or generation, should not be defined as part of the agenda's priority function, but rather as specific strategies.

In summary, $\mathcal{UTA}$'s properties 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.*

5 Item Sharing between Parsing and Generation

We will now present a new 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 to be performed in isolation, then such a method would seem to be an overhead. However, in the next section we will demonstrate that a strong interleaving of parsing and generation is a necessary prerequisite for modelling high-level performance strategies.

5.1 The basic idea

Assume that $\mathcal{UTA}$ is in parsing mode. Then each time a passive item is computed it is automatically also made available for the generation mode. Thus, for example, if we are going to generate from the semantics of the parsed input, we can directly return the previously computed answer during parsing as a result of the generation mode (i.e., if we only consider one paraphrase). Moreover, if we perform generation using a different semantics as the "parsed" one, but which is identical with respect to some partial semantic structures (e.g., some arguments are semantically identical with respect to the "parsed" semantics), then the generator also can "reuse" these partial results determined through parsing. Clearly, this kind of processing only makes sense if the same grammar and the same basic processing strategy are used during parsing and generation.

The restriction of sharing only passive items is plausible for the following reasons: Assume we are still in the parsing mode. Then, by means of the definition of item sets, the appropriate value for the index slot for the generation mode can be directly

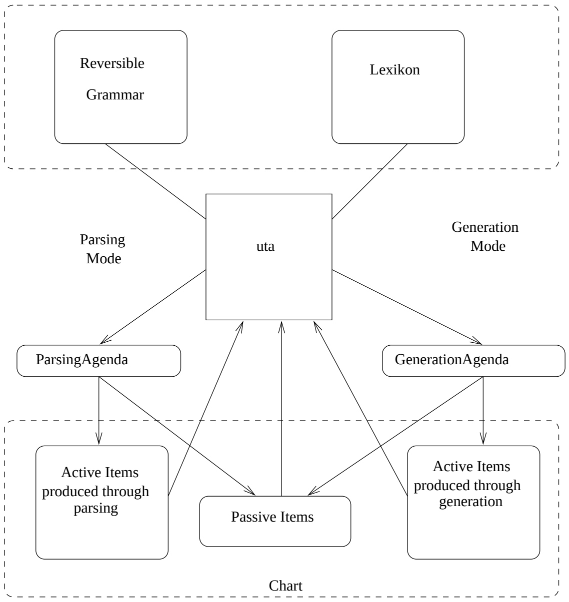

Figure 2: The *item sharing* approach of $\mathcal{UTA}$: During the different modes $\mathcal{UTA}$ maintains different agendas and private active items. However, passive items are shared by both directions.

determined on the basis of the semantic information of the passive item. This guarantees that shared passive items produced during the parsing mode are in the right place when they are used by the generation mode. On the other hand, the selected elements chosen for active items during parsing and generation will, in general, be different, and the essential feature of the other direction will be un-instantiated. Therefore, it would not be possible to determine the right place of a shared active item as is the case for shared passive items.

On the basis of these observations, the structure of an item sharing approach for $\mathcal{UTA}$ is as follows: We assume that $\mathcal{UTA}$ 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 item sets are considered as equivalence classes that are determined on the basis of the value of the essential feature, 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 both parser and generator have their own individual active items, but can operate on the same set of passive items. Figure 2 illustrates the structure of the item sharing approach.

5.2 Adaptation of the uniform tabular algorithm

In order to adapt $\mathcal{UTA}$ for the item sharing method the structure of an item is extended so that it contains different index slots *idx* for parsing and generation. Thus we have

$$\langle L;\ i;\ idx_p;\ idx_g\rangle$$

where L denotes the lemma of an item, i the (position of the) selected element. During parsing the slot idx_p is used and during generation the slot idx_g.

If we are in one of the two possible directions, say parsing, then for active items only the corresponding slot idx_p is filled with the current value of the essential feature. The slot idx_g is unbound, which will be denoted by using the symbol *none.* We will use the notation *phon(x)* to denote the value of the essential feature used during parsing and *sem(x)* to denote the value of the essential feature used during generation. The general structure of active and passive items is as follows. In the case of parsing, active items are of the form

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

and for passive items we have

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

where al is an active lemma with the selected element at position i in the body of al, $phon(i)$ is the index of the item set, al is a member. pl is a passive item with no selected element, and m the pointer to the head of the passive lemma. In the case of passive items, the values of the essential feature 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

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

and for passive items we have

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

The inference rules can easily be adapted to handle such item structures. Firstly, $\mathcal{UTA}$ only considers one index slot, depending on the major mode, for example idx_p for parsing. If a new re-solved lemma is active, only idx_p receives the value of the essential feature. The value of the generation slot idx_g is by default *none.* However, if a passive lemma pl is re-solved then the slot idx_g also receives a value determined on the basis of the essential feature specified for this direction (i.e.,the value of the $\langle \textsc{sem} \rangle$ path). This will simultaneously cause the creation of an empty "generation" item set with the corresponding index idx_g. If the agenda mechanism selects pl for insertion into idx_p at some point, then pl is simultaneously and destructively inserted into idx_g, or in other words, pl points into idx_p as well as into idx_g.

If we change the direction mode from parsing to generation and a new passive item pl_g is computed, then before pl_g is inserted into the agenda, we check whether it is a shared item by applying the blocking test. If this is the case, then pl_g is not added to the agenda, since we know that it is already in the chart. This sort of processing is of advantage if we use different preference strategies during parsing and generation, since it pretends that shared items will influence the determination of the preference values of next tasks.

5.3 An object-oriented extension of $\mathcal{UTA}$

In order to assign the correct value to the *idx* slot, $\mathcal{UTA}$ has to know in which mode it is. To make this an automatic task $\mathcal{UTA}$ 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 will automatically choose the right slot. The structure of the class PROOF is as follows:[12]

```
(defclass proof ()
 (name result-list agenda task-counter prio-fct))
```

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

```
(make-instance 'parse                  (make-instance 'generate
 :name "Parsing direction"              :name "Generation direction"
 :agenda (make-agenda)                  :agenda (make-agenda)
 :task-counter 0                        :task-counter 0
 :prio-fct #'depth-first)               :prio-fct #'depth-first)
```

All functions (with the few exceptions given below) are defined as methods for the class PROOF. This means that $\mathcal{UTA}$ only exists once, but is used by the two different instances. The advantage of using two different instances is that we can easily maintain different agendas or use specific priority functions for both instances. Thus, our implementation directly mirrors the architecture of the item sharing approach as shown in figure 2.

The only functions that are defined as specific methods for the parsing and generation classes are MAKE-ITEM and ADD-ITEM, and they differ only with respect to one additional call for a function. For the case of MAKE-ITEM, if a new lemma is passive, we have to determine values for the slots of the direction that is currently not active.

[12] The object-oriented extension of $\mathcal{UTA}$ has been implemented in CLOS, the Common Lisp Object System [Steele, 1990]. We therefore make direct use of the CLOS-class definition, abbreviated where convenient.

And for the case of ADD-ITEM we additionally have to add the new item to the corresponding item set (which has been created through 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 it.

For an illustration of the item sharing approach, see the parsing and generation example given in Appendix A (as well as figures 3 and 4). For example, when the passive item for the partial string "mit Maria" is re-solved during parsing (in figure 3 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 item 13 is:

$$15\langle pp;\ \epsilon;\ mM;\ m(M)\rangle 13$$

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

6 A Case Study: Incremental self-monitoring with reversible grammars

In this section we will demonstrate how interleaved parsing and generation based on $\mathcal{UTA}$ and item sharing is used for realising an incremental self-monitoring strategy. The core idea here is that during generation already produced partial strings are parsed to determine the degree of their 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 results in an utterance which is as unambiguous as possible.

The new approach is based on and an improvement of a non-incremental method presented in [Neumann and van Noord, 1992; Neumann and van Noord, 1994]. The basic scientific motivation of this work can be summarized 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.

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**.

In order to handle this problem the above cited articles present a mechanism which ensures that only non-ambiguous utterances are produced. This mechanism uses the parsing component to monitor the generation component. The relevant communication between the two components is performed using derivation trees. The underlying strategy is based on a comparison of the derivation trees obtained through generation and parsing, where the 'parsed trees' are computed with the output string of the generator. These parsed trees are used as a 'guide' for re-generating the utterance: If the parser yields several readings then each parsed tree is compared with the generation tree from the top downwards. When an ambiguous subtree is detected the generator is called with the semantics found at the root node of this subtree. This mechanism only makes sense for systems in which a single grammar is used for both parsing and generation.[13]

The problem of non-incremental monitoring The major drawback of the non-incremental method is that monitoring of a generated string only takes place *after* the whole string has been computed—thus its degree of interleaving is restricted. However, a fundamental assumption of this non-incremental version 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 already on a phrasal level, as illustrated by the following example:

(4) 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

(5) 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

(6) Visiting relatives are boring.

[13] We have also described a variant of the monitoring strategy which can be used to paraphrase a given input sentence (for interactive disambiguation) [Neumann and van Noord, 1994]. In this case, the generation component is used to guide the parsing system. Again the proposed technique is possible only in the case of a single, reversible grammar.

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 restricted on the phrasal level but also occurs 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

(7) 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

(8) I know of no better ball.

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

The need for contextual sensitivy The examples illustrate that a single 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 with respect to its context will remain unambiguous if it is part of a larger utterance. It seems 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 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 $vp \leftarrow v, np, pp$. 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., $np \leftarrow np, pp$ 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 $vp \leftarrow v, pp, np$ 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 looks back to already produced substrings, in order to test whether a new string together with previous produced substrings causes ambiguity.

6.1 Adaptation of $\mathcal{UTA}$ for performing incremental self-monitoring

We will now show how $\mathcal{UTA}$ is extended for realizing the incremental monitoring strategy, i.e., we will describe

1. how the context is determined and used for locating potential ambiguities, and
2. how revision is realized by $\mathcal{UTA}$.

The core idea is to define the ambiguity check as a function which decides whether a just created item should be added to the agenda or not. As shown in the example above, the ambiguity check should only take place for a newly generated substring before it is embedded into a larger structure. For $\mathcal{UTA}$ this means that it will take place if there is a passive item which can be used for reducing an active one. Hence, the ambiguity check should be defined as a further conditional statement of the completion rules for deciding that reduction should only be performed if revision is not requested. For example, the *passive completion* rule is changed as follows (only the relevant parts are expressed explicitly):

p-completion(Pi) is:
  For every active item $Ai \in I_{idx}$:
   if $\Phi = $ UNIFY(SEL(Ai), h) **and** $\Phi \neq fail$ **then**
    if NOT(AND ($Monitor?$,AMBIGUITY-CHECK($\Phi[Ai]$,Pi))) **then**
     with reduced lemma $Rl = \Phi[Ai - \text{SEL}(Ai)]$ **do**
     ...
   od fi.

We have added a new condition 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 and if no revision should take place, which is decided by the function AMBIGUITY-CHECK.[14] In the same manner *active completion* and *scanning* are modified.

Before we describe in detail how the ambiguity check is realized, we will show how revision is realized automatically by $\mathcal{UTA}$.

6.1.1 Performing revision within $\mathcal{UTA}$

From the revised version of *passive completion*, we can see that an item is not added to the agenda if the ambiguity check returns true, meaning that an ambiguity source has been detected. However, this implies that the search space has been actually cut for those branches where the item could be a subgoal. By means of this "built-in"

[14]Using a globally set flag to trigger incremental monitoring is 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 is generated without monitoring. We have actually *implemented* this any-time strategy.

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, it is checked whether p can reduce an active item a by means of passive completion. Then, before a is actually reduced using p, it is checked whether p causes an ambiguity using an appropriate context.

Only if no ambiguity can be determined, is the reduction of a performed and the resulting new item added to the agenda. On the other hand, 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 one sense, this kind of processing means that the selected element has been marked implicitly, 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 will automatically 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 would hence reuse results of previously made computations. If this is the case, the marker of p has been implicitly 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 to clarify the strategy. We are assuming the following simple grammar:

1. $s \leftarrow np,\ vp$
2. $vp \leftarrow v,\ np,\ pp$
3. $vp \leftarrow v,\ pp,\ np$
4. $np \leftarrow det,\ n$
5. $np \leftarrow np,\ pp$
6. $pp \leftarrow prep,\ np$

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, assumably without ambiguity problems. Next the *np* should be used for reduction. Before that, 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 is then used for reduction of rule 1.

In a similar way, other types of structural ambiguities are handled, e.g., ambiguities caused by the scope of negation, coordination or extraposition. For example, the ambiguity of the expression “Die kleinen Männer und Frauen” (*The small men and women*) could be solved by the paraphrase “Die kleinen Männer und die Frauen” (*The small men and the women*) or by “Frauen und die kleinen Männer” (*Women and the small men*). Similarily, in a sentence like “Den Studenten hat der Professor benotet, der das Programm entwickelte.” (*The-ACC student-ACC has the professor-NOM graded, who developed the program.*) the ambiguity caused by the extraposed *np* can be solved by means of “Den Studenten, der das Programm entwickelte hat der Professor benotet.” (*The-ACC student-ACC, who developed the program has the professor-NOM graded.*). For the extraposition example, however, the domain of locality is spanned by the whole sentence. In that case it would also be possible – and probably more efficient – to solve the ambiguity by producing a second sentence like *The student wrote the program* (see also sec. 7).

6.1.2 Performing ambiguity checks within $\mathcal{UTA}$

We now divert our attention to the problem of testing whether or not a new produced partial string causes ambiguity using the following definition:

ambiguity-check(Ai, Pi) is:
 with $ExtendedString$ = GET-CONTEXT(Ai,Pi,n);
 if $ExtendedString$ **then**
  with $ParsedResult$ = PARSE($ExtendedString$);
  if AND($ParsedResult$,AMBIGUOUS(Ai,$ParsedResult$))
   then true else false fi
 else false fi.

The function GET-CONTEXT determines the contextual information. If so, the parser is called with the extended-string built inside GET-CONTEXT, using a look-back of n. Revision should only take place if the parser obtained one or more readings successfully and if the result is ambiguous. Note the way $\mathcal{UTA}$ 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. It is just a side effect of $\mathcal{UTA}$ by not creating items which could cause ambiguity problems. Furthermore, because monitoring is applied on intermediate results, it is actually performed incrementally.

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 contains α. Such a larger string will be built by means of concatenation of α and some other string which has already been produced. We will call it the *contextual string* of α.

The ambiguity check 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. 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 with the derivation tree of the head of the lemma of the active item.[15] 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

$$\langle vp \leftarrow pp;\ 0;\ \textsc{vef}(0, Ef)\rangle$$

At that point the derivation tree represented as part of the constraints of *vp* is:

$$\begin{bmatrix} \texttt{rn} & \textit{vp3} \\ \texttt{phon} & \langle \textit{remove}, \textit{the}, \textit{folder}\rangle\text{-}P \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \left\langle \begin{bmatrix} \texttt{rn} & \textit{v5} \\ \texttt{phon} & \langle \textit{remove}\rangle\text{-}P1 \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \langle\ \rangle \end{bmatrix}, \begin{bmatrix} \texttt{rn} & \textit{np3} \\ \texttt{phon} & \langle \textit{the}, \textit{folder}\rangle\text{-}P2 \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \text{``\textit{its tree}''} \end{bmatrix}, Tree \right\rangle \end{bmatrix}$$

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 value of the variable $Tree$ in the above derivation tree is:

$$\begin{bmatrix} \texttt{rn} & \textit{pp1} \\ \texttt{phon} & \langle \textit{with}, \textit{the}, \textit{tools}\rangle\text{-}P \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \text{``\textit{its tree}''} \end{bmatrix}$$

We take this representation as the basis for the determination of the contextual string of the *pp*'s string "with the tools" making use of the *look-back* strategy as already informally described above.

[15] We assume that derivation trees are represented as part of the head's constraints of each rule and lexical element using the feature DERIV. The internal structure of this feature consists of the features LABEL the value of which is a constant that uniquely identifies this clause, and DTRS the value of which is a list of the derivation trees of the elements of the body of a clause. Additionally, two features PHON and SEM are used as pointers to the string and semantics of the clause, and as an interface for parsing and generation. We are using this representation, as reduction causes the removal of the completed elements from the body of a clause, so the elements of the body cannot be used directly.

A look-back(n) strategy We call the value of the DTRS feature the *sequence of sisters* of the node represented by the clause's head element. Since we consider the sister nodes as totally ordered, a look-back(1) strategy is the choice of the selected element's 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 the 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 contains the string of pp. Thus we can directly start parsing this string to test whether it is ambiguous.

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 rule. It might happend, e.g., that the pp is completed before the np is. Then we would have no (left) sister to be use-able as the contextual string for the pp, because the derivation tree of the np still needs to be constructed. If this is the case, we conclude that no statement about ambiguity can be made for the pp, and therefore, no revision should take place. After the np has been completed it will eventually be monitored. But now, it can choose its left or right sister as the base of contextual information, or both.

We can directly generalise the informal description to a look-back(n) strategy, if we choose 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 than n possible sister nodes to the left or right of the selected element.

The first case means that there is a sister whose derivation tree has not yet 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 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 from i to j.[16] 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 the derivation tree of each element of the subsequence is instantiated.

[16] Empty productions will be handled so that if a sequence contains the name of an empty production we simply 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 equal.

The look-back(n) strategy can now 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**.

The definition of GET-CONTEXT can now be given as follows:

get-context(Ai, Pi, n) is:
  with $Dtrs$ = GET-DTRS(Ai);
  with $Lsisters$ = GET-LEFT-SISTERS($Dtrs$,LABEL(Pi));
  with $Rsisters$ = GET-RIGHT-SISTERS($Dtrs$,LABEL(Pi));
    "apply look-back(n) on Lsisters and Rsisters;"
  if $ExtendedString$ **then**
   $ExtendedString$ **else false fi**.

Firstly, we extract the sisters of the derivation tree of the active item Ai, i.e., the value of the path $\langle deriv, dtrs \rangle$ of the constraints of the active item's lemma's head. Then we split this list into a left and right subsequence, where the passive item (which corresponds to the unified selected element of Ai) serves as the splitting point. Next, we apply the look-back(n) strategy, and either return an extended string or **false**, if no such string exists.

Ambiguity check Next we call the parser (i.e., we run $\mathcal{UTA}$ in the parsing mode), whose task is to parse the extended string. If it cannot be parsed, we conclude that no revision is necessary, and the ambiguity check 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 the definiton of which is as follows:

ambiguous$(ParsedResult, Ai)$ is:
  with $ReducedResult$ = "delete spurious ambiguities";
  if CARD($ReducedResult$) > 1 **then**

true else
 if SEM(*ReducedResult*) = SEM(*Ai*) **then**
  false else true fi fi.

First we delete all spurious ambiguities, i.e., for a pair of derivation trees which have the same semantics we only retain one.[17] 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 of the new string should take place. The former case means that the extended string has been analysed as unambiguous (since we have obtained only one result), but 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 hand, the semantic expression is not equal to that of the active item, we have found a possible ambiguity, and hence, revision should take place.

Degree of resolved ambiguity 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. 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.

Suppose we are following a look-back(1) strategy and that we have two grammars G_1 and G_2, which are weakly equivalent. The maximal branching factor of G_1 is 2 and that of G_2 is some integer m greater than 2. For G_1 a look-back(1) strategy means that in each case where the incremental monitor mechanism is activated, the newly determined extended string is identical with the whole string of the constituent defined by the active 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, it is possible, that not all ambiguities will be detected. Consequently, 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.[18] The discussion directly reveals

[17] The test for spurious ambiguity thus serves as a filter. Clearly, the current formulation of the test might be too simple. However, in principle it is not difficult to exchange it with a more complex test as long as the semantic representation of the grammar would support application of such a test.

[18] Of course, if we want to make sure that our algorithm behaves in the same way for grammars with a 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.

the problem of determining the appropriate value for the look-back strategy. If we choose the maximal branching factor, then we obtain unambiguous strings for the price of high computational effort. On the other hand, 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 have actually considered all relevant ambiguities.

In order to reach a 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. Assuming we have such criteria, they can easily be used during monitoring, so that GET-CONTEXT can use this information to decide whether an ambiguity check should take place for the passive item, and then the tests are applied on each sister "consumed" by the look-back strategy. Only if the passive item and its sisters satisfy the conditions expressed by these criteria will an extended string eventually be delivered. This provides the possibility to restrict the application of the monitoring strategy to grammatical information. As an example, it would be possible to restrict monitoring to maximal projections or to those structures which are known to cause ambiguities (e.g., pp-modifiers, coordinations). Our current implementation already provides mechanisms that can take into account such additional grammar specific information. However, it is a matter for future investigation (primarily on the linguistic side) to achieve meaningful and realistic criteria.

6.2 Using shared items during incremental monitoring

The main advantage of the incremental method using $\mathcal{UTA}$ described so far is that we benefit from the use of the chart during the monitored generation strategy, because in that case we can reuse previously made computations. Since revision is automatically performed by the agenda mechanism of $\mathcal{UTA}$ (by not creating items for those structures where an ambiguity has been detected), parsing of the extended string is the most costly additional operation during incremental monitoring. We will now demostrate how the incremental monitoring method can be rendered more efficient by making use of the *item sharing* approach described in 5.

Remember that in the item sharing approach passive items that have been computed in one direction can be used directly in the other. Following the method described in section 5, $\mathcal{UTA}$ maintains different agendas, item sets and active items for the parsing and generation mode, but passive items are shared by both directions. The object-oriented realization of the item sharing approach allows the parser 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, passive items are continually made available for the other direction. For example, for the interleaved parsing mode this means that it can reuse results computed through generation when performing the ambiguity check. During this job, however, it can provide results which can be used by the generator.

This means that in an interleaved mode parsing results are used through generation, and generation results are used through parsing.

7 Discussion and Related Work

7.1 Properties of the incremental monitor

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

More specifically, the incremental monitoring method can be seen as an additional restriction to $\mathcal{UTA}$ to keep track only of those partially computed 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 $\mathcal{UTA}$ 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.

If an un-ambiguous string cannot be found, this is due to the fact that some locally detected ambiguity could not be re-solved. In principle this problem could be handled by producing the ambiguous string followed by a separate paraphrase, which "explains" the local ambiguity. However, this implies that during incremental monitoring we would explicitly keep track of the points in a derivation in which a revision of a substring was requested. Clearly it is not difficult to extend the incremental monitor to behave in this way. Hence this mechanism could also be used as a more general strategy in which the generation of paraphrases is not only performed by the reformulation of previously generated substrings. Instead, revision would be delayed until the ambiguous sentence has been generated and the task of the incremental monitor would just be to mark the location of found ambiguity sources explicitly and to use it as an additional control information for the generation of following sentences.

7.2 Related work

$\mathcal{UTA}$ can be seen as extension of Shieber's uniform algorithm. It uses a dynamic selection function (where Shieber only uses the left-most selection function for both directions), and a truly uniform indexing mechanism (while Shieber only handles indices efficiently during parsing). Gerdemann [1991] also presents an extension of Shieber's algorithm that tries to make efficient use of indexing during generation. However, his degree of uniformity is restricted, since he actually uses different indexing mechanisms for parsing and generation.

$\mathcal{UTA}$ has a stronger goal-directed behaviour than the semantic head-driven algorithm described in [Shieber *et al.*, 1990], because it uses a semantic-oriented selection for all rules of the grammar (where Shieber et al consider only a subset of the rules; all other rules are processed in a simple left-to-right top-down manner). Furthermore, they do not make use of a chart. Van Noord [1993] has also extended this algorithm for head-corner parsing. One of the main problems with his approach is that it does not support incremental processing.

The use of the essential feature Ef as the single parameter of $\mathcal{UTA}$ is comparable to Strzalkowski's essential argument approach [Strzalkowski, 1994]. However, he only uses this information off-line during grammar compilation in order to obtain specific parsing and generation grammars. In [Erbach, 1995] a uniform algorithm based on bottom-up Earley deduction is presented that makes use of a flexible indexing scheme, mainly for the use of parsing however. In [Johnson and Dörre, 1995] an Earley deduction mechanism is presented using a mechanism which is able to coroutine between goals that depend on each others' partial solutions. However, they only consider parsing. Den [1994] presents a chart-based algorithm based on Earley deduction which uses an agenda mechanism similar to $\mathcal{UTA}$'s, in particular he presents a cost-based abduction method used to choose between alternative derivations. However, he too only considers parsing. Recently, Kay [1996] has presented a similar but more efficient indexing schema for chart-based generation. It would be worthwhile to adapt his indexing schema also to our uniform framework (cf. [Neumann, 1997]).

None of the above mentioned approaches use shared items, since they do not consider interleaved parsing and generation. Interleaved approaches can be found in the areas of artificial intelligence or cognitive science, e.g., [Jameson and Wahlster, 1982], [Vaughan and McDonald, 1986], [DeSmedt and Kempen, 1987], [Meteer and Shaked, 1988], [Levelt, 1989], [Wahlster *et al.*, 1991]. None of them however perform interleaving of parsing and generation with a comparable degree of granularity, nor do they consider uniform processing and item sharing.

8 Conclusion

We have developed a uniform computational model for natural language parsing and generation. It is based on $\mathcal{UTA}$, 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 and practical interleaving of parsing and generation is achieved by a novel incremental monitoring algorithm that is used during natural language production.

In the future we will explore methods which improve $\mathcal{UTA}$'s performance and we will investigate new interleaved strategies in the context of dialog systems. In order to improve $\mathcal{UTA}$'s efficiency we will explore explanation-based learning (EBL) as a method for the automatic extraction of subgrammars for controlling and speeding up uniform processing. In [Neumann, 1994a; Neumann, 1997] we already demonstrated the application of EBL to efficient parsing and generation of constraint-based grammars. We have now started to combine EBL-based generation and parsing to one *uniform* EBL approach. In particular we expect to improve the item sharing method to a kind of *template sharing* approach. Concerning dialog systems we will explore the integration of the incremental monitor with preference-based strategies and the interleaved approach in the context of clarification dialogs as part of a uniform question-answering system.

References

[Adriaens and Macken, 1995] Geert Adriaens and Lieve Macken. Technological evaluation of a controlled language application: Precision, recall and convergence tests for SECC. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 123–141, 1995.

[Alshawi and Crouch, 1992] H. Alshawi and R. Crouch. Monotonic semantic interpretation. In *30th Annual Meeting of the Association for Computational Linguistics*, Newark, Delaware, 1992.

[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.

[Barnett, 1994] J. Barnett. Bi-directional preferences. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 201–234. Kluwer, 1994.

[Berg, 1986] T. Berg. The problems of language control: Editing, monitoring and feedback. *Psychological Research*, 48:133–144, 1986.

[Bresnan, 1982] J. Bresnan, editor. *The Mental Representation of Grammatical Relations*. MIT Press, 1982.

[Carnegie Group, 1994] Carnegie Group. Carnegie Group's ClearCheck. *Language Industry Monitor*, (19 (Jan.-Feb.)):9–10, 1994.

[Cline and Nutter, 1994] B. E. Cline and J. T. Nutter. Kalos – a system for natural language generation with revision. In *AAAI-94*, pages 767–772, Seattle, Washington, 1994.

[Copestake *et al.*, 1996] A. Copestake, D. Flickinger, R. Malouf, S. Riehemann, and I. Sag. Translation using minimal recursion semantics. In *Proceedings, 6th International Conference on Theoretical and Methodological Issues in Machine Translation*, 1996.

[Den, 1994] Y. Den. Generalized chart algorithm: An efficient procedure for cost-based abduction. In *32nd Annual Meeting of the Association for Computational Linguistics*, New Mexico, 1994.

[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.

[Earley, 1970] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[Erbach, 1995] Gregor Erbach. *Bottom-Up Earley Deduction for Preference-Driven Natural Language Processing*. PhD thesis, Universität des Saarlandes, Germany, draft version, August 1995.

[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.

[Gerdemann, 1991] D. D. Gerdemann. *Parsing and Generation of Unification Grammars*. PhD thesis, University of Illinois, Cognitive Science, Technical Report CS-91-06, 1991.

[Haas, 1989] A. Haas. A parsing algorithm for unification grammars. *Computational Linguistics*, 15(4):219–232, 1989.

[Höhfeld and Smolka, 1988] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical Report 53, LILOG IBM, Stuttgart, 1988.

[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 the 5th European Conference on Artificial Intelligence*, pages 222–227, Orsay, France, 1982.

[Johnson and Dörre, 1995] M. Johnson and J. Dörre. Memoization of coroutined constraints. In *33rd Annual Meeting of the Association for Computational Linguistics*, Cambridge, 1995.

[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, 1996] M. Kay. Chart generation. In *34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, Ca., 1996.

[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.

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

[Levine, 1992] J. M. Levine. Pragma: A flexible bidirectional dialogue system. In *AAAI-90*, pages 964–969, Boston, 1992.

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Programming*. Symbol Computation, Springer, Berlin, New York, 1987.

[McKeown *et al.*, 1997] K. R. McKeown, D. A. Jordan, S. Pan, J. Saw, and B. A. Allen. Language generation for multimedia healthcare briefings. In *5th International Conference of Applied Natural Language*, pages 277–282, Washington, USA, March 1997.

[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.

[Nerbonne, 1992] J. Nerbonne. Constraint-based semantics. Research Report RR-92-18, DFKI GmbH, Kaiserslautern und Saarbrücken, BR Deutschland, 1992.

[Neumann and van Noord, 1992] Günter Neumann and Gertjan 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, 1994a] 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.

[Neumann, 1994b] G. Neumann. *A Uniform Computational Model for Natural Language Parsing and Generation*. PhD thesis, Universität des Saarlandes, Germany, November 1994.

[Neumann, 1997] G. Neumann. Applying explanation-based learning to control and speeding-up natural language generation. In *35th Annual Meeting of the Association for Computational Linguistics/8th Conference of the European Chapter of the Association for Computational Linguistics*, Madrid, Spain, 1997.

[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, 1994] C. Pollard and I. M. Sag. *Head-Driven Phrase Structure Grammar*. Center for the Study of Language and Information Stanford, 1994.

[Ristad, 1993] E. S. Ristad. *The Language Complexity Game*. MIT-Press, 1993.

[Robertson, 1994] S. P. Robertson. Tsunami: Simultaneous understanding, answering, and memory interactions for questions. *Cognitive Science*, 18:51–85, 1994.

[Samuelsson, 1994] C. Samuelsson. *Fast Natural-Language Parsing Using Explanation-Based Learning*. PhD thesis, Swedish Institute of Computer Science, Kista, Sweden, 1994.

[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.*, 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, Stanford University, SRI International Technical note 460, 1989.

[Shieber, 1993] S. M. Shieber. The problem of logical-form equivalence. *Computational Linguistics*, 19:179–190, 1993.

[Smolka, 1992] G. Smolka. Feature constraint logics for unification grammars. *The Journal of Logic Programming*, 12:51–87, 1992.

[Somers *et al.*, 1990] H. Somers, J. Tsujii, and D. Jones. Machine translation without a source text. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, volume 3, pages 271–276, Helsinki, 1990.

[Steele, 1990] G. L. Steele. *Common LISP: The Language (Second Edition)*. Digital Press, Burlington, MA, 1990.

[Strzalkowski, 1994] T. Strzalkowski. A general computational method for grammar inversion. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 175–199. Kluwer, 1994.

[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.

[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 *et al.*, 1991] W. Wahlster, E. André, W. Graf, and T. Rist. Designing illustrated texts: How language production is influenced by graphics generation. In *5th Conference of the European Chapter of the Association for Computational Linguistics*, pages 8–14, Berlin, 1991.

[Wahlster, 1991] W. Wahlster. User and discourse models for multimodal communication. In *Intelligent user interfaces*, chapter 3, pages 45–67. ACM Press, 1991.

[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.

A A Complete Parsing and Generation Example Run with $\mathcal{UTA}$

We will use the following grammar fragment to illustrate the behaviour of $\mathcal{UTA}$:[19]

$$
\begin{array}{l}
\textsc{vp}(\textsc{Sem}) \leftarrow \textsc{v}(\textsc{Sem}),\ \ \textsc{np},\ \ \textsc{pp} \\
\textsc{vp}(\textsc{Sem}) \leftarrow \textsc{v}(\textsc{Sem}),\ \ \textsc{pp},\ \ \textsc{np} \\
\textsc{vp}(\textsc{Sem}) \leftarrow \textsc{v}(\textsc{Sem}),\ \ \textsc{np} \\
\textsc{np}(\textsc{Sem}) \leftarrow \textsc{n}(\textsc{Sem}) \\
\textsc{np}(\textsc{Sem}) \leftarrow \textsc{np}(\textsc{Sem}),\ \ \textsc{pp} \\
\textsc{pp}(\textsc{Sem}) \leftarrow \textsc{p}(\textsc{Sem}),\ \ \textsc{np}
\end{array}
$$

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 Ef is bound to $\langle$PHON DL$\rangle$ and for generation the value is bound to $\langle$SEM$\rangle$.

Figure 3 illustrates how $\mathcal{UTA}$ processes the string "sieht Peter mit Maria" ("sees Peter with Mary") during its parsing mode, and figure 4 shows the trace of the semantic expression "sehen(Peter,mit(Maria))" ("to_see(Peter,with(Mary))"). The simple grammar used 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 analysed and for the reading "sehen(peter, mit(maria))" the two strings "sieht Peter mit Maria" and "sieht mit Maria Peter" are generated. Thus the example illustrates very well how we can reuse completed structures in parsing as well as in generation.

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

$22\langle ans; \epsilon; 0\rangle 22$
$21\langle vp; \epsilon; 0\rangle 21$
$1\langle vp \leftarrow v, np, pp; 0; 0\rangle 18$
$18\langle ans; \epsilon; 0\rangle 16$
$17\langle vp; \epsilon; 0\rangle 15$
$2\langle vp \leftarrow v, np; 0; 0\rangle 2$
$0\langle ans \leftarrow vp; 0; 0\rangle 1$

sPmM_0

$19\langle vp \leftarrow np, pp; 0; 1\rangle 19$
$16\langle np; \epsilon; 1\rangle 14$
$8\langle np; \epsilon; 1\rangle 7$
$7\langle np \leftarrow n; 0; 1\rangle 6$
$4\langle np \leftarrow np, pp; 0; 1\rangle 5$
$5\langle np \leftarrow n; 0; 1\rangle 4$
$3\langle vp \leftarrow np; 0; 1\rangle 3$

PmM_1

$20\langle vp \leftarrow pp; 0; 2\rangle 20$
$15\langle pp; \epsilon; 2\rangle 13$
$10\langle pp \leftarrow p, np; 0; 2\rangle 9$
$9\langle np \leftarrow pp; 0; 2\rangle 8$

mM_2

$12\langle np \leftarrow np, pp; 0; 3\rangle 17$
$14\langle np; \epsilon; 3\rangle 12$
$13\langle np \leftarrow n; 0; 3\rangle 11$
$11\langle pp \leftarrow np; 0; 3\rangle 10$

M_3

Agenda	CurrentTask	Item of alternative
0	0	$1\langle vp \leftarrow v, np, pp; 0; 0\rangle$
1,2	2	
1,3	3	
1,4,5	5	$4\langle np \leftarrow np, pp; 0; 1\rangle$
1,4	4	
1,6,7	7	$6\langle np \leftarrow np, pp; 0; 1\rangle$
1,6,8	8	
1,6,9	9	
1,6,10	10	
1,6,11	11	
1,6,12,13	13	$12\langle np \leftarrow np, pp; 0; 3\rangle$
1,6,12,14	14	
1,6,12,15	15	
1,6,12,16	16	
1,6,12,17	17	
1,6,12,18	18	First Result
1,6,12	12	
1,6	6	
1	1	
19	19	
20	20	
21	21	
22	22	Second Result
ϵ		

Figure 3: A trace through parsing of the string "sieht Peter mit Maria".

We assume that a lemma counter is used that enumerates just created lemmas (starting from 0) and that the agenda mechanism selects tasks in a depth-first manner. The items that have been placed in any item set are also counted starting by 1. The lemma counter will be attached to an item as a prefix, and the item counter as its suffix. To make things more readable, we use 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 "sPmM_0". 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, the current selected task and those items which represent alternatives. The latter are displayed in an extra row "Item of alternative", to make the depth-first strategy more readable.

$19\langle ans; \epsilon; 0\rangle 20$
$18\langle vp; \epsilon; 0\rangle 19$
$1\langle vp \leftarrow v, pp, np; 0; 0\rangle 16$
$15\langle ans; \epsilon; 0\rangle 13$
$14\langle vp; \epsilon; 0\rangle 12$
$2\langle vp \leftarrow v, np, pp; 0; 0\rangle 2$
$0\langle ans \leftarrow vp; 0; 0\rangle 1$

$\mathrm{s(P,m(M))}_0$

$17\langle vp \leftarrow np; 0; 1\rangle 18$
$4\langle np \leftarrow np, pp; 0; 1\rangle 15$
$6\langle np; \epsilon; 1\rangle 5$
$5\langle np \leftarrow n; 0; 1\rangle 4$
$3\langle vp \leftarrow np, pp; 0; 1\rangle 3$

P_1

$16\langle vp \leftarrow pp, np; 0; 2\rangle 17$
$13\langle pp; \epsilon; 2\rangle 11$
$8\langle pp \leftarrow p, np; 0; 2\rangle 7$
$7\langle vp \leftarrow pp; 0; 2\rangle 6$

$\mathrm{m(M)}_2$

$10\langle np \leftarrow np, pp; 0; 3\rangle 14$
$12\langle np; \epsilon; 3\rangle 10$
$11\langle np \leftarrow n; 0; 3\rangle 9$
$9\langle pp \leftarrow np; 0; 3\rangle 8$

M_3

Agenda	Current Task	Item of alternative
0	0	
1,2	2	$1\langle vp \leftarrow v, pp, np; 0; 0\rangle$
1,3	3	
1,4,5	5	$4\langle np \leftarrow np, pp; 0; 1\rangle$
1,4,6	6	
1,4,7	7	
1,4,8	8	
1,4,9	9	
1,4,10,11	11	$10\langle np \leftarrow np, pp; 0; 3\rangle$
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: A trace through generation of "sehen(Peter,mit(Maria))".

We use the abbreviations introduced in figure 3. 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. Note that we need to use the path $\langle$SEM$\rangle$ as the essential feature. This is the only requirement to let $\mathcal{UTA}$ run for generation in an efficient manner. The selection function "simulates" the semantic-head first selection function, although coincidentally in all cases the head element is located in the leftmost position. The second paraphrase is generated by reusing the PP "mit Maria" (item 13) and the NP "Peter" (item 6) is 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 sequence 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.

List of Figures