THE INTEGRATED DEDUCTIVE APPROACH TO NATURAL LANGUAGE INTERFACES

eingereicht von:

MMag. Werner Winiwarter

DISSERTATION

zur Erlangung des akademischen Grades

Doctorum rerum socialium oeconomicarumque
(Doktor der Sozial- und Wirtschaftswissenschaften)

Sozial- und Wirtschaftswissenschaftliche Fakultät der Universität Wien

Betreuung:

o.Univ.-Prof. Dipl.-Ing. Dr. A Min Tjoa
o.Univ.-Prof. Dr. Günther Vinek

Wien, im Juni 1994

THE INTEGRATED DEDUCTIVE APPROACH TO NATURAL LANGUAGE INTERFACES

by

MMag. Werner Winiwarter

Submitted for the Degree of Doctor of Economic Sciences

Institute of Applied Computer Science and Information Systems

Department of Information Engineering

School of Social and Economic Sciences, University of Vienna, Austria

June 1994 - Vienna, Austria

Abstract

The framework of our research originates from two different sources: *natural language processing* and *deductive database technology*. Deductive databases possess superior functionality in comparison to relational systems relevant to the efficient solution of many problems arising in practical applications, yet there still exists no broad acquaintance and acceptance. As main obstacle we identified the absence of any user-friendly interface. *Natural language interfaces* have been proposed as optimal candidate for complex database applications because they make it possible to communicate with the database system without the need to learn any formal query or manipulation language. However, in spite of the vast number of ambitious attempts to build natural language front-ends, the achieved results as concerns user acceptance were rather disappointing. In our opinion there are two main reasons for this: *missing customisation*, resulting in unexpected restrictions, and *missing integration*, responsible for insufficient performance and wrong interpretation. We deal with these shortcomings by combining the strengths of both research fields. In our *Integrated Deductive Approach (IDA)* the interface constitutes an integral part of the database system itself by making use of the declarative power of deductive databases, that is, the dictionary of the interface is also designed as component of the database system as well as the complete natural language analysis is performed by the logic programming language provided by deductive databases. This *complete integration of linguistic analysis* guarantees the consistent mapping from the semantic representation of the user query to the appropriate semantic application model avoiding any discontinuities of homogeneity. For each step of natural language analysis we introduce new concepts and show their efficient implementation in IDA. *Morphological and lexical analysis* are performed by adapting the *lexical approach*, also covering prefixes, derivations, and compound words. The dictionary has a *hierarchical structure* making it possible to insert all features at the appropriate level of abstraction. Furthermore, the expressive *two-level formalism* is applied to the processing of three special morphological phenomena: ablaut, elision, and binding sounds. As concerns *syntactic analysis*, we propose an extension to *Categorial Unification Grammar* in order to analyse free word order languages efficiently. This grammatical framework is in optimal conformity with the powerful dictionary as well as the evaluation strategy of deductive databases. The *bottom-up parsing* also makes it possible to analyse *incomplete and ungrammatical sentences* in an easy and natural way. For *semantic analysis* we introduce the *unknown value list (UVL) analysis*, a technique that operates directly on the evaluation of database values and deep forms of functional words, that is, syntactic analysis is only applied if necessary for disambiguation. Finally, also solutions for *discourse resolution* and *spelling error correction* are presented. We prove the feasibility of the IDA approach by use of a *case study*, the design and implementation of a *production planning and control system*. The central point of our proposed *seven step model* to the development of efficient database applications with natural language interfaces is the *empirical collection of test data* in order to obtain realistic input sentences, therefore guaranteeing optimal customisation for practical use.

Zusammenfassung

Der theoretische Rahmen dieser Arbeit hat seinen Ursprung in zwei Bereichen: *Verarbeitung natürlicher Sprache* und *deduktive Datenbanktechnologie*. Obwohl deduktive Datenbanken im Vergleich zu relationalen Systemen über eine erweiterte Funktionalität verfügen, welche für die effiziente Bewältigung zahlreicher in Praxisanwendungen auftretender Probleme von Relevanz ist, war das bisherige allgemeine Interesse sowie die Benutzerakzeptanz gering. Da hierbei das hauptsächliche Hindernis die fehlende benutzerfreundliche Schnittstelle darstellt, wurde als optimaler Kandidat eine *natürlichsprachliche Schnittstelle* vorgeschlagen. Trotz der großen Anzahl an ambitionierten Versuchen, natürlichsprachliche Oberflächen zu entwickeln, waren die erzielten Resultate in Hinblick auf die Benutzerakzeptanz enttäuschend, wofür zwei Hauptfaktoren verantwortlich gemacht werden können: *fehlende Anpassung,* resultierend in unerwarteten Restriktionen, sowie *fehlende Integration,* welche unbefriedigendes Systemverhalten und unkorrekte Interpretationen verursacht. In dieser Arbeit werden diese Unzulänglichkeiten durch die Kombination der Stärken beider Forschungsbereiche beseitigt. Im *Integrierten Deduktiven Ansatz (IDA)* stellt die Schnittstelle einen Teil des Datenbanksystems selbst dar, sodaß das Lexikon als Komponente des Datenbanksystems entworfen sowie die vollständige natürlichsprachliche Analyse mittels der logischen Programmiersprache der deduktiven Datenbank realisiert wird. Diese *vollständige Integration der linguistischen Analyse* garantiert die konsistente Abbildung der semantischen Repräsentation der Benutzerabfrage auf das entsprechende semantische Anwendungsmodell. Für jeden Analyseschritt werden neue Konzepte eingeführt und deren effiziente Implementierung in IDA gezeigt. *Morphologische und lexikalische Analyse* werden unter Adaptierung des *lexikalischen Ansatzes* durchgeführt, wobei auch Präfixe, Derivative und Komposita berücksichtigt werden. Das Lexikon besitzt eine *hierarchische Struktur,* welche es ermöglicht, alle Informationen auf der geeigneten Abstraktionsebene einzutragen. Darüberhinaus wird der mächtige *Two-level-Formalismus* auf die Verarbeitung dreier spezifischer morphologischer Phänomene angewendet: Ablaut, Elision und Bindelaute. In bezug auf die *syntaktische Analyse* wird eine Erweiterung zur *Kategorialen Unifikationsgrammatik* vorgeschlagen, um Sprachen mit freier Wortstellung effizient analysieren zu können. Diese Grammatiktheorie ist in optimaler Übereinstimmung sowohl mit dem mächtigen Lexikon als auch mit der Evaluierungsstrategie deduktiver Datenbanken. Die *Bottom-up-Strategie* ermöglicht es auch, *unvollständige und ungrammatikalische Sätze* auf einfache und natürliche Art und Weise zu analysieren. Für die *semantische Analyse* wird die *Analyse unbekannter Werte* eingeführt, eine Methode, die direkt auf der Evaluierung der Datenbankwerte und Tiefenformen der Funktionswörter aufsetzt, sodaß die syntaktische Analyse nur eingesetzt wird, falls sie für die Disambiguierung notwendig ist. Schließlich werden auch Lösungen für *Discourse Resolution* und *Eingabefehlerkorrektur* präsentiert. Der vorgestellte Ansatz wird auf eine *Fallstudie* angewendet, die Implementierung eines *Produktionsplanungs- und -steuerungssystems.* Die zentrale Komponente des *Sieben-Schritt-Modells* für die Entwicklung effizienter Datenbankanwendungen mit natürlichsprachlichen Schnittstellen ist die *empirische Ermittlung von Testdaten,* welche die optimale Anpassung an den praktischen Einsatz gewährleistet.

The Integrated Deductive Approach to Natural Language Interfaces

Werner WINIWARTER

Institute of Applied Computer Science and Information Systems
Department of Information Engineering
University of Vienna, Austria

1. Motivation

Deductive database technology emerged during the past decade, it combines the strengths of both *logic programming* and *relational database algebra*. The extended functionality led the way to solutions to practical problems which could not be handled efficiently before. In spite of the superiority in comparison with *relational database systems*, there still exists no broad acquaintance and acceptance with regard to practical applications [Lockemann92]. Since we identified the *user interface* as the main obstacle for a specific user to become familiar with a new database paradigm, our objective was to supplement deductive databases with a user-friendly front-end.

Starting from the first days of research on *natural language processing*, the use of unrestricted language has been regarded as optimal choice to the communication of casual users with sophisticated database applications. The great advantage of natural language is based on the fact that the user is not forced to learn any *formal query or manipulation language* (see Figure 1). Furthermore, it possesses more expressive power and flexibility than other user-friendly interfaces, e.g. graphical front-ends or menu-based systems.

Intensive work was done during the last decades and a huge number of prototypes were developed but somehow they suffered the same fate as deductive databases: they are still far away from widespread practical use [Copestake90]. The reason for this are the many limitations that still exist and which are caused by two main factors: missing *integration* and missing *customisation*. We deal with both problems by introducing on the one hand a new type of architecture, the *Integrated Deductive Approach (IDA)*, on the other hand a *seven step model* to the development of a fully customised database application with natural language interface.

IDA brings together the two 'fellow sufferers' natural language interfaces and deductive databases in that the *interface constitutes an integral part of the database system itself*. This signifies that the complete natural language analysis is performed by the powerful logic language supplied by deductive databases which guarantees for the first time a homogenous mapping of the semantic representation of user input to the underlying database application [Winiwarter93a].

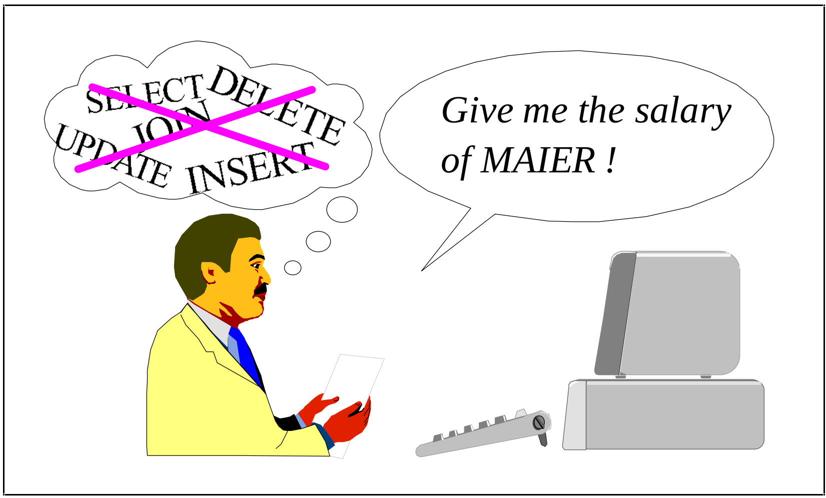

Figure 1: Natural language interface

The proposed *seven step model* to the development of database applications with natural language interfaces has as central step the *empirical collection of test data* by use of questionnaires. Only this procedure guarantees the complete coverage of all relevant linguistic phenomena which is essential for full customisation and achievement of wide user acceptance in later practical use.

As introduction to this thesis we start in *Chapter 2. Introduction to Natural Language Processing* with some basic definitions and concepts before we give a short survey of the current state of research on natural language processing. Special emphasis is given to existent work on *natural language interfaces*, other mentioned recent approaches refer to application areas like *information retrieval, information filtering* or *machine translation.* The second part of the introduction is supplied by *Chapter 3. Deductive Databases* in which first some fundamentals and indications about existent work are stated. For our research we use as implementation platform the prototype system *SALAD* which was developed at MCC on the basis of the deductive database language *LDL,* an extension of the computational paradigm *Datalog*. Therefore, we give an insight into the syntax and functionality as far as necessary for the understanding of this thesis.

The central part of this work introduces new concepts and methods for each step of natural analysis within IDA (for the reference language German) and illustrates how these concepts can be implemented efficiently in LDL. In *Chapter 4. Morphological and Lexical Analysis* the initial two steps of natural language analysis are dealt with because they are carried out at the same time in IDA by adapting the *lexical approach* which stores only canonical forms and assigns all features to the dictionary entries. We develop the required formal framework consisting of *morpho-syntax* and *two-level rules* which is able to handle prefixes, derived words, and compound words as well as the special morphological phenomena *ablaut, elision,* and *binding sounds*. In *Chapter 5. Syntactic Analysis* we discuss existent grammar formalisms as basis for selecting *Categorial Unification Grammar (CUG)* as framework for the syntactic analysis in IDA. Since CUG as such is not well-suited for the analysis of free word order languages, we introduce some important extensions which make it possible to analyse also incomplete and ungrammatical sentences in a natural and efficient way. In *Chapter 6. Semantic and Pragmatic Analysis* we present a semantic analysis technique based on *unknown value list (UVL) analysis* that makes optimal use of the information supplied by the *semantic application model* of the underlying database application and uses syntactic analysis only for disambiguation of several interpretations. Finally, we propose an efficient *discourse resolution method* and a new *similarity measure* for correcting misspelled database values.

The final part of this thesis aims at proving the feasibility of the previously introduced new concepts. In *Chapter 7. Case Study: PPC Database for Precision Tools* we report on a case study: the design and implementation of a production planning and control system (PPC) with German natural language interface for an enterprise which manufactures precision tools. By applying the proposed *seven step model* we illustrate the individual steps of the design and development process in full detail. Based on the extensive test data, the concluding evaluation examines the correct functionality and performance, in particular the achieved response times.

2. Introduction to Natural Language Processing

2.1 Basic Concepts

Natural language processing encompasses all computer-based approaches to the handling of unrestricted written or spoken language [Lunin84], the latter also referred to as *speech processing*. This definition is much more general than that of *natural language understanding*. Whereas the latter is always concerned with the underlying meaning, natural language processing also deals with simple methods applied in word processors, indexing procedures or keyword-matching retrieval techniques.

With regard to the direction of computation, natural language processing can be divided in two general processes:

- *natural language analysis:* the natural language input is mapped to some internal representation
- *natural language generation:* the internal representation is transformed to natural language output

The complex challenge of natural language analysis is usually divided in sub-tasks by applying the general process model displayed in Figure 2.

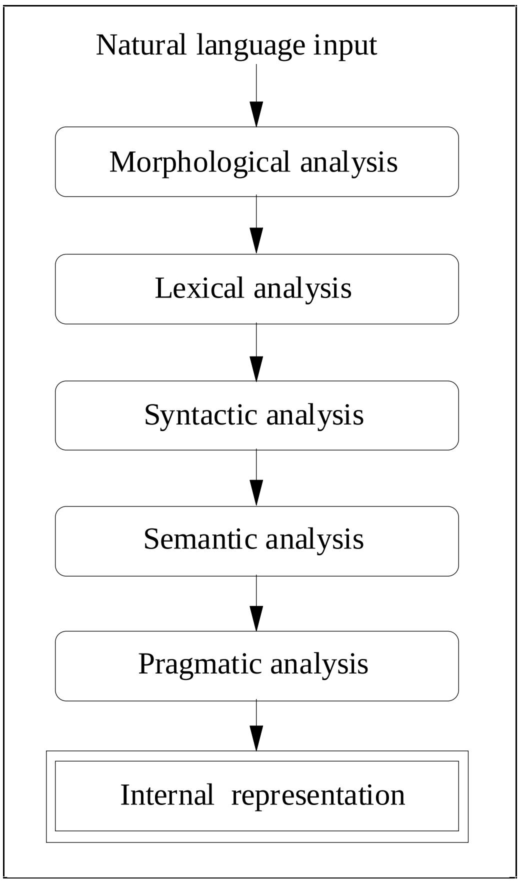

Figure 2: Process model of natural language analysis

The individual steps of the process model perform the following computations, they need not be looked upon as strictly sequential but are often realised in an interleaved manner [Doszkocs86]:

- *morphological analysis:* by means of lemmatisation each word is transformed to its canonical form, the removed endings supply syntactic information like number, person or gender
- *lexical analysis:* the canonical forms are looked up in the dictionary retrieving the associated syntactic and semantic information
- *syntactic analysis:* by use of a parser the syntactic structure of a sentence is generated according to the defined grammar
- *semantic analysis:* based on the sentence structure a semantic representation is determined which covers the intended meaning
- *pragmatic analysis:* the scope of examination is extended from the analysis of a single input sentence to the interpretation of the whole recent context of discourse by making also use of some kind of world knowledge

If one considers also spoken language, new problems arise like word recognition and segmentation (for recent work on this research field see [White90, Dowding93, Rowles93, Nagao93b]). On the other hand, supplementary prosodic information is available like pitch and pause information [Cruttenden86, Beler93, Raskutti93a]. A second class of recent approaches deals with the additional complexity of recognising hand-written sentences [Keenan91, Hull92, Srihari93].

Whereas the main difficulty of natural language analysis is to *disambiguate* several possible interpretations of a specific input sentence, for the *generation* procedure the opposite is true, that is, one must choose one of the many possible surface representations, e.g. according to user modelling [Sparck-Jones91]. For good surveys of the research on generation see [McDonald87, Reiter93], recent approaches include applications to *response creation* in interfaces [Kalita86, Chu93, Raskutti93b], *explanation systems* [Moore91, Paris91, Zukerman93, Suthers93, McKeown93] or *automatic documentation* [Kittredge86, Hovy91, Mittal93].

Natural language processing has been dominated for over three decades by the *knowledge-based approach.* Only recently, an 'empirical renaissance' takes place for domain-independent applications which is mainly caused by the following three reasons (see [Church93]):

- huge data collections like *machine-readable dictionaries* or *text corpora* are now available
- advances in computer technology have resulted in a significant cost reduction of powerful hardware architectures
- a greater emphasis on deliverables and evaluation exists for which the corpora supply a uniform test-bed

The many recently proposed techniques are mostly statistical methods [Briscoe93, Brown93, Smadja93, Weischedel93, Magerman94] but also neural networks are regarded as promising tool for the use in natural language analysis [Jain91, Daelemans92, Jacquemin93, Merkl94b].

2.2 Natural Language Interfaces

Generally speaking, natural language interfaces enable the easy data access without using any formal commands but by simply defining the request in plain English or any other natural language. There exist three different interface types (for general surveys see [Bates87, Capindale90, Copestake90]):

- interfaces to knowledge bases
- interfaces to information retrieval systems
- interfaces to databases

Interfaces to knowledge bases have been usually implemented as integrated logical or functional programming systems. They often were not realised as interfaces to existent applications but on the contrary formed the central issue of the system, such realisations are referred to as *natural language (understanding) systems* or *question-answering systems*. Prominent examples are SHRDLU [Winograd72], Chat-80 [Warren82], VIE-LANG [Zsolnai87], MURPHY [Selfridge86] or WISBER [Fliegner88]. The main drawback of these systems are the missing features of database management systems like transaction support, schema-based integrity or efficient secondary storage.

The second category of interfaces accesses unstructured information stored in *information retrieval systems* (see [Salton83]). In contrast to database interfaces the complex part of these interfaces is not the analysis of the input sentence but that of matching the text documents with the query in order to produce a satisfactory query result. Because of the huge size of information retrieval systems in practical use, natural language analysis is often combined with simpler techniques like Boolean search in order to reduce the number of documents to be analysed. Examples for successful implementations are Iota [Chiaramella87], I^3R [Croft87], Adrenal [Lewis89] or Amics [Caraceni93].

Most existent *natural language database interfaces* deal with the access to relational database systems: the early systems RENDEZVOUS [Codd74] and PLANES [Waltz77, Waltz78] as well as more recent systems like TEAM [Grosz82, Grosz83], SESAME [Ali86] or System X [McFetridge88]. Also for German language some promising prototypes have been developed, e.g. PLIDIS [Berry-Rogghe78], HAM-ANS [Höppner83] or Datenbank-DIALOG [Trost90].

The crucial weak point from what all these systems suffer is the mapping from the final semantic representation of the input sentence to the actual database query which incorporates a discontinuity of homogeneity as concerns the different semantic models (e.g. mapping of relations or attributes, see [Schröder88] for a detailed discussion).

The second great difficulty that database interfaces differently from other natural language applications must cope with is the processing of database values as part of the user query. As especially systems that claim to be domain-independent do not access the knowledge contained in the database for use in natural language analysis, the usual approach is to assume that undefined words represent database values (see [McFetridge90]). Therefore, if one considers the possibility of misspelled values, they are not able to distinguish between new database values for insertion or update and misspelled existent data.

The very first database interface LUNAR [Woods72] as well as the first commercially available natural language interface INTELLECT [Harris84] tried to overcome this situation by retrieving the concerned values from the database. However, due to the huge search spaces and the limitations of relational database technology, this method severely affects the efficiency of the application.

A different approach to the resolution of unknown values is to restrict the complexity of input resulting in some kind of *pseudo-natural language*, examples of such systems are LADDER [Sacerdoti77, Hendrix78], TQA [Damerau81], ENLI [Kambayashi86, El-Sharkawi90] or HAVANE [Bosc86]. Some implementations even delimit the use of natural language to a menu-based system, e.g. NL-MENU [Tennant83, Thompson83] (see also [Rich87]). This decrease of complexity guarantees an efficient analysis but also leads to a significant reduction of habitability which questions the main reason for using natural language instead of formal query languages [Ogden87].

Although deductive databases combine the advantages of the first and third type of interfaces, that is, the integrated architecture and the database management facilities, therefore incorporating the power to solve all those shortcomings, there is no existent work that makes full use of this power. The only known prototype of a natural language interface to a deductive database was designed by Gal/Minker who focused their research on the generation of natural language answers to user queries [Gal85]. However, also this prototype uses only a loosely coupled interface resulting again in the above mentioned inaccuracies for the treatment of unknown words.

2.3 Other Applications

Beginning from the first days of natural language processing, *text* or *document analysis* has represented one of the central research fields. The main differences to the design of natural language interfaces are the large vocabulary, the complete and correct nature of sentences, the enriched complexity of applied sentence structures and the larger scope of context meaning.

An important application area is *information retrieval* where the semantic representations of documents are used for the match with user queries and the presentation of the query result. The process of the creation of these representations is also often referred to as *knowledge acquisition*. For examples of approaches that apply linguistic analysis for that purpose see [Dillon83, Smeaton86, Katz88, Antonacci89, Schwarz90].

The great shortcoming of all these systems is their high development cost which makes them not feasible for large-scale applications. This is especially true for the field of legal information retrieval systems (see [Schweighofer93a]). Therefore, again neural networks [Belew87, Bench-Capon93, Merkl94a] as well as statistical techniques [Wong87, Salton88, Schweighofer93b, Schweighofer94] have been proposed as efficient empirical solutions. Another challenging application area is multimedia data, there exists some recent work that faces its specific needs, in particular the semantic representation of graphical, video or audio data [Baudin93, Arens93, Han93].

Closely related to the applications in information retrieval are *information filtering systems* which have as main task the distribution and delivery of information in order to assist the user in finding relevant information [Höfferer94d]. Early work was concerned with summarising and classifying messages for very specific domain ranges, e.g. ATRANS [Lytinen84] or TESS [Young85] for the analysis of banking messages. More recent work deals with emails, NetNews articles, newswire stories or multimedia documents (see [Höfferer94a]).

Sundheim [Sundheim92] established with the *message understanding conference* an international forum for the objective evaluation of information filtering systems with regard to the parameters speed, precision, and recall [Chinchor92]. Two examples of successfully evaluated prototypes are SCISOR and FASTUS.

The system SCISOR was designed by Jacobs/Rau [Jacobs90] for the analysis of financial news taken from the on-line financial service of Dow Jones. SCISOR employs a combined bottom-up and top-down natural language analysis by means of a knowledge base for conceptual representation. To improve the performance irrelevant messages are discarded at an early stage of analysis by a powerful filter topic analyser. The prototype achieved within the MUC-II evaluation both 90 % of precision and recall while analysing 6 messages per minute.

FASTUS was implemented at SRI International [Hobbs92]. Instead of parsing the text by use of a context-free grammar it applies a non-deterministic finite-state language model which decomposes a sentence into noun groups, verb groups, and particles. The extraction of relevant phrases is activated by means of trigger words and the individual retrieved information parts are finally merged to obtain a unified representation for the whole text. In spite of its simple architecture FASTUS achieved within the MUC-4 evaluation a precision of 55 % and a recall of 44 %, the number of analysed words per minute was 2375, that is, approximately also six stories are analysed per minute.

Whereas all evaluated systems still possess only static behaviour, already first prototypes of adaptive systems were proposed [Höfferer94b] which take into consideration the user behaviour by use of a monitor component [Höfferer94c].

Another interesting application of natural language processing is the use of natural language as input for the design of databases. The user specifies the requirements in natural language sentences which are mapped to corresponding concepts in a conceptual data model. As important feature, the user is kept aware of actual integrity conflicts and is asked for correction, examples of recent implementations are ISTDA [Bracchi83], MODELLER [Tauzovich89] or DMG [Tjoa93].

Finally, we want to conclude this short survey by perhaps the most classical use of natural language processing which was right from the beginning always connected with high expectations but could seldom satisfy them: *machine translation*. The first part of processing is in analogy to that of interfaces, that is, to compute the meaning of the input sentence. Based on this internal representation, the output sentence is generated in the target language (for a good introduction see [Schwanke91]). Without the use of an internal representation, for each pair of languages a separate translation module would be required. Only the use of a language-independent semantic representation scheme makes it possible to apply an efficient star-shaped architecture. Prominent commercial systems for German are LOGOS [Wheeler86], SUZY [Luckhardt82], SYSTRAN [Wagner85] or METAL [Gebruers88].

The recent 'rebirth' of empirical natural language processing also strongly stimulated the research on machine translation. Among the many current approaches applying statistical methods are [Katz89, Brown90, Dorr90, Kitano93, Sumita93, Phillips93, Frederking93]. The most famous and ambitious recent research activity is carried out by ATR in Japan which deals with the automatic translation of telephone calls (see [Black93, Matsumoto93]).

3. Deductive Databases

3.1 Introduction

The theory of deductive databases has been the topic of intensive research within the last years (for good reviews see [Gallaire84, Ullman89, Ullman90]). It has its origins in *logic programming* and *relational database algebra*. By combining the advantages of both research fields, deductive databases provide the following important extensions with regard to functionality [Naqvi89]:

- there exists the possibility of formulating *recursive queries*, that is, transitive relationships can be considered
- the nonmonotonic operation of *negation* is supported
- not only atomic object types but also *complex object types* like sets, trees or lists can be used for data modelling
- *updates* are performed by means of declarative specifications
- *imperative predicates* are available which enable the use of conventional control structures, the declarative semantics is preserved

In spite of this superiority which is not only of scientific interest but on the contrary possesses significant relevance for solving many problems in practice, there still exists no broad acquaintance and acceptance for deductive databases [Lockemann92].

As concerns the first attempts to build prototypes of deductive databases, two main approaches were followed:

- The coupling of Prolog and relational database systems, e.g. [Kunifji84, Li84, Bocca86, Chang86, Cuppens86, Jarke86, Ceri87b, Ioannidis87, Lefebre89]. These prototypes were motivated by the immediate availability of Prolog and relational database technology but suffer from a number of drawbacks so that they cannot support the required level of functionality, performance, and ease of use [Zaniolo90a].

- The strong integration of the expressive power of logic programming and the performance and robustness of relational database systems by implementing the minimal fixpoint semantics of rule sets and the underlying relational data. For a general view of the architectural aspects of these systems see [Zaniolo90b], for examples of implementations we refer to *Section 3.2 Datalog.*

The most influential theoretical framework for the second category of approaches is the computational paradigm *Datalog* which we will describe in *Section 3.2 Datalog* in more detail. Datalog has formed the basis for several extensions and successful implementations. One of the most prominent is *LDL* (Logical Data Language) which was designed and implemented at MCC as portable prototype system for UNIX called *SALAD* (System for Advanced Logical Applications on Data) [Naqvi89, Chimenti90].

Since we used SALAD as implementation platform for our research, we present in *Section 3.3 LDL* the basic concepts and constructs of LDL which are necessary for the comprehension of the various examples we will provide throughout this work. Finally, *Section 3.4 SALAD* gives some additional details about the architecture of SALAD and the applied compilation techniques.

3.2 Datalog

Datalog represents a rule-based computational paradigm which was specifically designed for the purpose of interacting with large relational database systems (for a complete survey see [Ceri89, Ceri90]) and which is based on a subset of general *Logic Programming* [Lloyd87].

A Datalog *program* is defined as a finite set of *Horn clauses* consisting of *literals* L_i:

$$L_o \leftarrow L_1, \ldots, L_n. \tag{3.1}$$

The left-hand side of a clause is called *head*, its right-hand side *body*. If the *body* is left empty, the clause represents a *fact*, otherwise it models a *rule*. A literal is written as *predicate* with a *predicate symbol* p_i and a number of *terms* t_i:

$$L_i = p_i(t_1, \ldots, t_k) \tag{3.2}$$

The terms t_i can be either *variables* or *constants*. In order to distinguish the two types, variables are capitalised whereas constants and predicate symbols have to start with lower-case letter.

A literal or clause without variables is called *ground*. Any Datalog program must satisfy two safety conditions in order to guarantee that the set of all facts which can be derived is finite:

- each fact is ground
- each variable occurring in the head of a rule must also occur in its body

Example 3.1:

angular_part(backe, aluminium, 2, 3, 5).	fact for attributes of angular part with constants: name, raw material, length, width, height
stock(backe, 5).	fact for stock of parts with constants: name, quantity
lagernd(eckteil, Name, Quantity) ← angular_part(Name, Material, L, W, H), stock(Name, Quantity).	rule which derives names and quantities for stock of angular parts

■

In accordance to the intention of Datalog and in contrast to general Logic Programming the facts and rules are not stored within a single logic program but are separated in two different sets:

- *Extensional Database (EDB):* the set of facts which is stored in the relational database
- *Intensional Database (IDB):* the set of rules and facts constituting the Datalog program

Analogously, the set of predicates is partitioned:

- *EDB-predicates:* the set of predicates which occur in the EDB
- *IDB-predicates:* the set of predicates which occur in the IDB but not in the EDB

Therefore, each EDB-predicate can be mapped to a corresponding *relation* in the relational database and each fact in EDB can be inserted as *tuple*. Similarly, the IDB-predicates can be regarded as *views*. Finally, *goals* perform the function of *queries*, they consist of a single literal preceded by a question mark, e.g. for *Example 3.1*: ?lagernd(eckteil, Name, Quantity).

A lot of work was done concerning the efficient evaluation of Datalog goals. The proposed algorithms can be roughly divided in two groups (for a detailed taxonomy see [Ceri90]):

- *evaluation methods* in which optimisation is performed during the evaluation itself, e.g. Gauss-Seidel method [Chang81, Bancilhon85], Semi-naive evaluation [Bancilhon86a, Ceri86], Henschen-Naqvi method [Henschen84] or query-subquery algorithm [Vieille86]
- *rewriting methods* which transform the program to a more efficient equivalent program before evaluation, e.g. magic sets [Bancilhon86b, Beeri87b], Counting [Bancilhon86b, Beeri87], magic counting [Sacca87a], static filtering [Kifer86] or Variable Reduction and Constant Reduction [Ceri87a]

By extending the very restricted Datalog syntax and by applying the above evaluation and rewriting methods some successful research prototypes have been implemented, e.g. *SALAD* (*Section 3.4 SALAD*), *NAIL!* [Morris86, Morris87], *KIWI* [Sacca87b] or *ALGRES* [Ceri88]. Also first attempts of combining Datalog with concepts from object-oriented databases exist [Beeri88, Czejdo88, Cacace89, Lee90, McCabe92, Ishikawa93] resulting in various *deductive object-oriented database system* prototypes like COMPLEX [Greco90], LLO [Lou91], LOL [Bertino92, Bertino93], OSAM*.KBMS [Su93] or CLOG [Hui93, Hui94].

3.3 LDL

LDL (Logical Data Language) was designed at MCC as purely declarative logic-based language. It provides many powerful extensions of pure Datalog which we will present in brief in the following (a complete presentation gives [Naqvi89], see also [Zaniolo85, Tsur86, Chimenti87, Beeri87b, Zaniolo90c]).

3.3.1 Data Types

The *arguments* (constants) used in *base predicates* (EDB-predicates) can possess three different *simple data types*: *string*, *integer,* and *real*. String constants which start with a capital letter have to be enclosed in single quotation marks in order to make them distinguishable from variables. The definition of the individual data types is performed within the *schema*.

Example 3.2:

For the two facts in *Example 3.1* the schema definition has the following form, the labelling of the arguments is optional:

```
ppc({   angular_part(Name: string, Material: string, Length: integer,
                Width: integer, Height: integer),
        stock(Name: string, Quantity: integer)}).
```
■

In addition to these three simple data types, arguments can also have nested structures resulting in *complex data types*.

Example 3.3:

The schema in *Example 3.2* is slightly changed in that the three dimensions are united:

```
ppc({   angular_part(Name: string, Material: string, Dim: (integer, integer, integer)),
        stock(Name: string, Quantity: integer)}).
```
■

Finally, there exist two special built-in complex data types: *lists* and *sets* [Shmueli88].

Example 3.4:

The first of the two following definitions defines operation sequences as list of actions, the second gives the set of machine types which an operator can handle:

```
ppc({ operation_sequence(Name: string, Actions: [string]),
      operator(Name: string, Qualification: {string})}).
```
■

3.3.2 Built-in Predicates

Built-in predicates are either written like other predicates or as special predicate symbols in infix notation. They are used like EDB-predicates (i.e. only in the body of rules) though they are not stored explicitly but evaluated during execution time. The following groups of built-in predicates can be distinguished:

- *comparison predicates:*
  - L = R
  - L ~= R $L \neq R$
  - L > R
  - L < R
  - L >= R $L \geq R$
  - L <= R $L \leq R$
- *arithmetic predicates:*
  - L + R
  - L - R
  - L * R
  - L / R
  - L mod R
- *list predicates:*
  - [X|Y]   X ... head   Y ... tail
- *set predicates:*
  - member(E, S) $E \in S$
  - subset(S1, S) $S1 \subseteq S$
  - union(S1, S2, S) $S = S1 \cup S2$
  - difference(S1, S2, S) $S = S1 - S2$
  - intersection(S1, S2, S) $S = S1 \cap S2$
  - cardinality(S, N) $N = |S|$

The *equality predicate* can also be used as *unification operation* if free variables are involved [Lassez88, Siekmann90]. As special case *single assignment* occurs if one side is a variable and the other is a constant term [Naqvi89].

There exists a special predicate for providing DON´T CARE non-determinism. As declarative equivalent of the CUT-operator in PROLOG, the *choice-predicate* in the rule

$$a(X, Y) \leftarrow b(X, Y), choice((X), (Y)) \quad (3.3)$$

creates a maximal subset of the predicate b(X,Y) under the preservation of the functional dependency X→Y [Krishnamurthy88a].

Example 3.5:

The following rule selects for each supplier one of the parts which he supplies:

liefer(Name, Part) ← supplier(Name, Part), choice((Name), (Part))

If only one supplier shall be retrieved randomly, then the rule has to be modified:

liefer2(Name) ← supplier(Name, Part), choice(( ), (Name))

In order to add to the one selected supplier one of his parts, the rule looks like this:

liefer3(Name, Part) ← supplier(Name, Part), choice(( ), (Name, Part)) ■

Finally, a predicate for *aggregation operations* on sets is provided, its internal representation has the following form [Naqvi89]:

```
aggregate(Op, S, V) ←
        if(empty(Op, V)   then true
                          else if(S = {X} then single(Op, X, V)
                                          else partition_once(S1, S2, S)),
                          aggregate(Op, S1, V1),
                          aggregate(Op, S2, V2),
                          multi(Op, V1, V2, V)).                    (3.4)
```

The predicate partition_once partitions the given set in two arbitrary disjoint subsets. Therefore, the aggregation operation examines recursively the empty, the singleton, and the general case.

Example 3.6:

The following aggregation operator menlist transforms a set to a corresponding list, the sequence of the list members is arbitrary. It uses the predicate append which joins two lists together.

```
empty(menlist, [ ]).
single(menlist, X, [X]).
multi(menlist, X1, X2, X) ← append(X1, X2, X).

append([X|Y], Z, W) ← append(Y, Z, W1), W = [X|W1].
append([ ], X, X).
```

■

3.3.3 Negation

Only predicates in rule bodies with *covered* variables can be negated (by use of the ~-symbol), that is, they get their value range from the evaluation of other positive predicates. The only exception are *existential* variables which only appear once in a rule (also *singleton* variables, normally written as *anonymous* variables by an underscore). A further restriction to the use of negation is that it cannot be applied in a recursive definition. By moving the negation out of the scope of the recursion one results in a *stratified program* [Chandra85, Przymusinski87, Naqvi87].

Example 3.7:

The following program decides if the number of list elements is even, it is non-stratified because of the use of negation in the recursive definition:

```
even([_|Rest]) ← ~even(Rest).
even([ ]).
```

To transform this program in a stratified version, the additional predicate `odd` is applied:

```
even(L) ← ~odd(L).
odd([_]).
odd([_|[_|Rest]]) ← odd(Rest).
```

■

3.3.4 Grouping

The *grouping* operator (<...>) collects several solutions of the evaluation of a variable into a unique set and can only be used in the head of rules [Beeri89]. In the same way as with negation one has to pay attention to stratification, that is, not to use the grouping operator in a recursive definition [Shmueli87].

Example 3.8:

The following rule computes the number of existent angular parts:

```
anzahl(Quantity) ←                     counts number of angular parts
    anzahl2(Set),                      set of angular parts
    cardinality(Set, Quantity).        cardinality of set equals number of angular parts
anzahl2(<Name>) ←                      grouping operator produces set of part names
    angular_part(Name, _, _, _, _).
```

■

3.3.5 Updates

The use of updates enriches the semantics of the logic program from first-order logic to dynamic logic [Ramakrishnan88] in the sense that the order of update specifications may affect the result of evaluation. All changes to the EDB-predicates are reduced to two opposite operations, the deletion (-) and insertion (+) of facts.

Example 3.9:

The following simple rule changes the stock for the base predicate stock from *Example 3.2*. If there does not exist a stock for the part in question, then no deletion is performed and the stock is added as new fact.

```
aendmen(Name, Quantity) ←
        -stock(Name, _),
        +stock(Name, Quantity).
```
■

By supporting the notion of database transactions [Sacca88a, Naqvi88] no failures are allowed after an update operation because this would signify that the update has to be revoked. Furthermore, this restriction makes it possible to detect the violation of integrity constraints (e.g. type mismatch) immediately. Only assignments, updates, and imperative predicates (see below) are *infallible predicates*, that is, they can be used after updates. Again, in order to guarantee stratification, updates cannot be used in recursive rules.

3.3.6 Imperative Predicates

The extension of LDL by update operations gives rise to the need for two imperative predicates [Naqvi88]. The first one of them is the *if-predicate*:

h ← if (p then q else w) (3.5)

which is semantically equivalent with the two rules:

h ← p, q.
h ← ~p, w. (3.6)

The common abbreviation:

h ← if (p then q) (3.7)

with the semantics:

h ← if(p then q else true) (3.8)

is also valid in LDL.

Besides the increase of legibility and evaluation efficiency, the if-predicate provides the essential possibility to make predicates infallible.

Example 3.10:

The following predicate updates a stock and applies a second predicate for delivery control which can fail. The left-hand side shows a wrong version in which the predicate is used as such after the update operations, the version on the right-hand side removes this error by the application of the if-predicate.

```
aktmen(Name, Quantity) ←
        -stock(Name, _),
        +stock(Name, Quantity),
        delivery_control(Name).
```

```
aktmen(Name, Quantity) ←
        -stock(Name, _),
        +stock(Name, Quantity),
        if(delivery_control(Name) then true).
```
■

The second imperative predicate is the iterative *forever-predicate* which is also infallible. Since updates cannot be used in recursive rules, the forever-predicate is used for such situations where an iterative definition of updates is needed.

The forever-predicate in a rule:

$$h \leftarrow g, \text{forever}(p), q. \quad (3.9)$$

is evaluated as:

$$H \leftarrow g, p_1, p_2, ..., p_n, q. \quad (3.10)$$

where the p_i are successive iterative applications of the p predicate. p_n is determined either by the fact that p_{n+1} fails or that $p_n = p_{n-1}$, that is, no further changes due to updates have occurred [Naqvi89]. Variable values can be imported into the forever-predicate but the only way of exporting evaluation results out of the forever-predicate is via updates.

Example 3.11:

The following rule shows the scheduling of the individual machining operations contained in a production list. As prerequisite the machining operations have been numbered before. Since the scheduling predicate itself hides a complicated planning and optimisation process resulting in many updates, e.g. for the assignment of workers and machines, the forever-predicate has to be applied in order to guarantee that each scheduling step is performed on the basis of the result of the prior planning decisions. For that purpose, a simple base predicate is used as loop counter, the evaluation exits the loop if the counter value exceeds the number of the last entry.

```
produktion(Plist) ←                     scheduling of production list
   +seqnr(1),                           initialising loop counter
   forever(                             for each machining operation in production list do
      seqnr(I),                            loop counter
      lmember((I, Operation), Plist),      retrieving actual machining operation
      plan(Operation),                     scheduling for individual machining operation
      J = I + 1,                           incrementing loop counter
      -seqnr(_),                           deleting old loop counter
      +seqnr(J)),                          storing new loop counter
   -seqnr(_).                           deleting loop counter
```
■

3.4 SALAD

The first LDL implementation used FAD, a language based on relational algebra supported by a massively parallel database machine [Danforth85, Boral88]. After its successful completion in 1987 at MCC, the next step was the design and implementation of an open architecture prototype system called SALAD (System for Advanced Logical Applications on Data) which was finished in 1988. SALAD operates within the UNIX environment and generates target code in C. It preserves the purely declarative semantics of LDL, e.g. in contrast to Prolog the order of rules is insignificant. Additionally, SALAD possesses the usual features of database management systems, i.e. support for transactions, recovery, schema-based integrity, and efficient management of secondary storage [Chimenti90].

The deductive database system consists of four main components which are strictly separated in four different file types (see Figure 3):

- a *schema* for base predicates: *.sch
- a set of *facts* representing the data (EDB-predicates): *.fac
- a set of *rules* for deriving new predicates (IDB-predicates): *.rul
- a set of *query forms* for generating access plans to stored data: *.qf

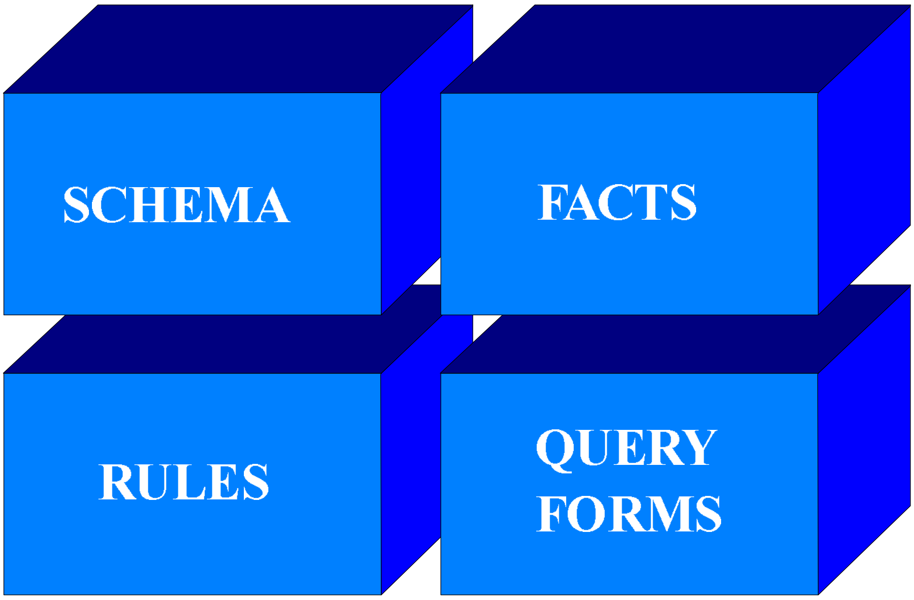

Figure 3: Components of SALAD

The query forms are *generic goals* in that they specify which arguments represent input parameters (covered variables, indicated by a preceding $-sign) and which are expected as output (free variables). These *bindings* are essential for the efficient compilation of the rule set [Zaniolo88]. The important difference to logic programming systems like Prolog is that in SALAD the facts are treated differently from the rules, they are described by the schema at compilation time. Therefore, any update can be performed freely without the need for recompiling or reinterpreting the program [Chimenti90]. Figure 4 shows a simple example for the file configuration in SALAD by use of the predicates from *Example 3.1*. The query form gives the user a list of all available stocks for a specific category of parts.

Two other important features of SALAD are modules and externals [Chimenti89b]. *Modules* allow for modular decomposition resulting in reduced target size code and possibly shorter execution time. The predicates used inside of a module are *local*, *global predicates* are defined via import and export specifications of query forms. *Externals* provide the essential possibility to write external predicates and functions in C (or FORTRAN) by the support of a powerful *external interface library*. For example, this library includes functions for the manipulation of lists and sets or for the access to base and global predicates [Chimenti89a].

STOCK.SCH

```
ppc({angular_part(Name: string, Material: string, Length:integer,
                  Width: integer, Height: integer),
     stock(Name: string, Quantity: integer)}).
```

STOCK.FAC

```
angular_part(backe, aluminium, 2, 3, 5).
stock(backe, 5).
```

STOCK.RUL

```
lagernd(eckteil, Name, Quantity) <-
      angular_part(Name, Material, L, W, H),
      stock(Name, Quantity).
```

STOCK.QF

```
qform lagernd($Category, Name, Quantity).
```

Figure 4: Example of SALAD files

Finally, the following extensions have been added [Chimenti89c]:

- composition of new data types
- declaration of indices and key constraints
- input/output primitives
- formatted output primitives
- input/output on files

The compilation of query forms is performed in several steps. First, the rules are rewritten by inserting the covered variables, a process called *constant migration*. Then, if this migration reaches base predicates, a corresponding selection is applied against the facts (*selection pushing)* [Krishnamurthy88b]. Recursive rules are compiled by use of *semi-naive fixpoint* [Sacca88a], *magic set method* [Sacca87c], and *generalized counting method* [Sacca88b].

With regard to the *execution mode* of the SALAD compiler, four different strategies are applied which are all in conformity with the bottom-up semantics of LDL [Chimenti90]. As illustrative example let

$$p(X, Z) \leftarrow a(X, Y), b(Y, Z). \qquad (3.11)$$

be a rule which is queried. Then, in terms of relational algebra, this query is answered by first computing the tuples resulting from the evaluation of the predicates a and b before the resulting tuples are joined over the common variable Y and projected on X and Z.

- *pipelined execution:* only those tuples in b are computed which join with tuples in a in a pipelined way (one at a time), if a tuple in b joins with several tuples in a, it is every time computed anew
- *lazy pipelined execution:* the tuples for b are stored in a *temporary relation* in order to avoid recomputations of the same tuple
- *lazy materialized execution:* differs from lazy pipelined execution in that for a given value of Y all joining tuples in b are computed and stored in a temporary relation before proceeding
- *materialized execution:* computes all tuples in b and stores them in a temporary relation before proceeding

The main difference between pipelined and materialized execution is that the former is favourable for backtracking whereas the latter is preferred for the use in recursion. The lazy variants only add computational overhead in order to improve the performance of the compiler [Chimenti89d].

The feasibility of the SALAD prototype has been tested by applying it to some problems of practical relevance. These problems reach from typical business applications like *processing of bills of materials*, *inventory control* or *job shop scheduling* [Tsur90a] to more advanced applications like *data dredging* (i.e. testing and formulating of hypotheses based on empirical data) [Tsur90b] or *scientific databases* [Tsur90c].

For the complex field of *enterprise modelling*, extensions to the Entity-Relationship model [Chen76] have been proposed and implemented as prototypes for CASE tools. The *POS (Process-Object-State) modelling technique* [Ackley90a, Ackley90b] considers integrity constraints, dynamic aspects, and aggregation and maps the specification to a deductive database application which automatically checks for inconsistencies and the violation of constraints.

Finally, there exists an application to the design of *MLS (multi level secure) database systems* (for more information about the MLS relational data model see [Jajodia91, Smith92]). The *Deductive Filter Approach* defines a *security constraints language (SCL)* for specifying application dependent constraints as LDL predicates [Pernul93a] as well as corresponding graphical extensions to the Entity-Relationship model [Pernul93b]. The specified constraints are checked in order to detect conflicting situations. Therefore, the resulting CASE tool guarantees a consistent conceptual representation of security semantics.

3.5 Summary

We shortly presented in this Section deductive database technology which is based on logic programming and relational database algebra. One of the most prominent deductive database languages is LDL which is a powerful extension of pure Datalog and was implemented in the prototype system SALAD at MCC.

The expressive power of logic programming, the support of complex object types, the possibility of using external C-predicates, the declarative semantics, and the neat separation of facts and rules, all these features made SALAD an ideal choice as implementation platform for the development of natural language interfaces in IDA architectures.

4. Morphological and Lexical Analysis

4.1 Introduction

In comparison with the vast amount of publications about the other components of natural language analysis, there exists only limited work concerning morphological issues. The main reason for this can be seen in the simplicity of English with regard to this respect whereas the treatment of morphological phenomena possesses a much higher significance for highly inflexional languages like German.

Simple approaches to morphological analysis deal only with the removal of endings and suffixes by means of a general pre-defined *suffix-tree* and do not take into account the proper analysis of prefixes and compound words [Thurmair82, Dorffner85]. Of course the number of words accepted by these general suffix-trees is much too voluminous (containing more invalid derivations than legal ones) so that the number of produced canonical forms must be reduced afterwards by use of additional information, like supposed word categories or ending classes added to the stem in the dictionary [Finkler88].

One further disadvantage besides this missing precision concerns the inherent syntactic and semantic information comprised in the removed endings. Although assignments of corresponding features are imaginable, the resulting semantic representation lacks flexibility to a high degree, e.g. there exists no possibility to deal with cases where a derived word gets a new specific meaning different from the word sense which the combination of the stem and the suffix in question would suggest.

To overcome these shortcomings the so-called *lexical approach* can be applied which assigns all morphological features directly to the corresponding canonical forms in the dictionary [Whitelock88]. Among the authors who contribute to that approach only few make full use of the available expressive power. With regard to retrieval efficiency, additional dictionary entries are often included for derivations by use of prefixes. However, this destroys the compact structure of the dictionary [Aoe90].

An important extension for achieving an efficient and natural representation of the syntactic and semantic features associated with morphological phenomena is the removal of the flatness of dictionaries by supplying them with a *hierarchical structure*. The individual features can then be assigned in a flexible manner to the appropriate level of abstraction [Smedt84]. By use of inheritance mechanisms the affixes are on the one hand supplied with general syntactic and semantic categories which on the other hand can be overwritten by information directly attached to the specific word derivations in order to express divergent connotations.

Two-level morphology has represented the most influential formalism within the last decade. It was developed by Koskenniemi for the Finnish language [Koskenniemi83]. This formalism introduces an additional surface level in order to deal with special morphological phenomena (e.g. vowel-gradation) in an elegant and compact style. It has been extended (e.g. [Karttunen87, Bear88]) and adapted to several other languages like Tamil [Sarkar93], French [Genikomsidis88] or German [Emele88].

4.2 Basic Concepts

In accordance with our intention of integrating the complete natural language analysis into the deductive database system by making full use of the declarative power of LDL, we adapted the *lexical approach* by storing only *canonical forms* in the dictionary and assigning to them all the *morphological features*, including also prefixes and compound words.

VERB(	
mess,	stem of *to measure*
...	
11,	conjugation class
2,	past participle class
{ab},	prefix *ab* yielding *to survey*
{(er,{[durch]}),	suffix *er* in combination with prefix *durch* yielding the noun *diameter*
(ung,{[]})}	suffix *ung* yielding the noun *measurement*
).	

Figure 5: Example of morphological features

Figure 5 shows a simple example of the assignment of morphological features to a verb. In addition to information about the conjugation of the verb, a set of possible prefixes can be declared which constitutes derived verbs. Finally, a set of suffixes together with sets of required prefix sequences can be defined for deriving nouns or adjectives. The dictionary entry shown in Figure 5 therefore covers all together 47 different surface forms (see Figure 6) including also irregular verb forms, compound verbs, and declensions of the derived nouns and of the adjectival use of both participles (by making use of auxiliary dictionary entries, see Figure 7).

messen, messe, miß, mißt, meßt, maß, maßest, maßen, maßet, messend, messender, messendem, messenden, messende, messendes, gemessen, gemessener, gemessenem, gemessenen, gemessene, gemessenes, abmessen, messe ab, miß ab, mißt ab, meßt ab, maß ab, maßest ab, maßen ab, maßet ab, abmessend, abmessender, abmessendem, abmessenden, abmessende, abmessendes, abgemessen, abgemessener, abgemessenem, abgemessenen, abgemessene, abgemessenes, durchmesser, durchmessern, durchmessers, messung, messungen

Figure 6: Example of coverage of surface forms

An input sentence is first separated into a list of single words by means of an external C-predicate. In the next step each individual word is compared with the dictionary entries whether the latter form proper sub-strings of it. Only if such an agreement is detected, the remaining parts of the input word are checked against the affixes recorded in the dictionary.

By use of the set data type of LDL it is also possible to represent ambiguities at the level of inflexions and affixes as well as at the word level in a consistent way.

```
VERBFORM(
            maß,     irregular verbform
            mess,    verb stem
            13,      conjugation class
            0        no past participle formed
                     from that verbform
        ).

VERBPRAEF(
            ab       separable verbprefix
        ).

SUBSTSUFFIX(
            ung,     suffix for deriving noun
            fem,     gender
            3        declination class
        ).
```

Figure 7: Example of auxiliary dictionary entries

Of course this sub-string test method is only feasible with regard to performance criteria for relatively small dictionaries. Therefore, it is very well suited for the use in database interfaces. For applications where such a narrow and well-defined universe of discourse does not exist (e.g. machine translation) other retrieval methods must be used, reducing again the transparency and conciseness of the dictionary [Aoe90].

As a consequence of the above mentioned advantages of a *hierarchically structured dictionary*, we supplied the flexible insertion of syntactic and semantic features at the appropriate level in the hierarchy and employed *inheritance mechanisms* for the analysis process. All properties are inherited from the ancestors unless more specific properties defined at a lower level overwrite more general attributes. Therefore, an efficient and natural representation is obtained, also taking into account divergent specific meanings of derived words.

Finally, to capture three morphological phenomena of particular relevance to the German language, namely *ablaut*, *elision*, and *binding sounds*, the *two-level formalism* is employed. The different rules are not generally valid but are restricted to the appropriate word classes, that is, they build an integral part of the hierarchically structured dictionary, again by deriving full benefit of the applied inheritance mechanisms [Emele88].

4.3 Morpho-Syntax

After the above general preliminary remarks we will now develop the underlying formal framework of our morphological analysis (see also [Winiwarter93b]). Of course our aim was not to obtain a complete representation of each morphological phenomenon which might ever occur in German but a reasonable and easily extendible set of rules which covers all cases relevant for the application in natural language interface design. The general *decomposition format* for lemmatising an input word is stated as follows, required parts are underlined:

CATEGORY = PREFIXPART <u>CATEGORY-STEM</u> SUFFIXPART (4.1)

4.3.1 Particles

They represent the linguistic units that are most easily analysed because they do not possess neither prefixes nor suffixes:

PARTICLE = ϕ <u>PARTICLE-STEM</u> ϕ (4.2)

Because of their irregular declensions we treated also articles, pronouns, and numerals as particles, that is, we stored the individual inflections as additional dictionary entries.

4.3.2 Nouns

German nouns are declinable with regard to case and number. Not only simple prefixes but also complex prefix lists can be put in front of the noun stem. The contents of the prefix lists is not restricted to prefixes in the usual sense but can also include other parts of speech for modelling compound words. Only complete prefix lists are accepted as input because there exist numerous cases where sub-lists constitute no legal word forms:

NOUN = [PREFIX] <u>N-STEM</u> ENDING (4.3)

Example 4.1:

Subteilhierarchien = [sub,teil] hierarchie n (sub-part hierarchies) ■

In addition to these regular situations nouns can also be derived from verbs, adjectives or other nouns by adding substantival suffixes to them:

NOUN = [PREFIX] <u>V-STEM | A-STEM | N-STEM</u> <u>SUFFIX</u> ENDING (4.4)

Example 4.2:

Mitarbeiter = [mit] arbeit er ϕ (to work -> employee)
Tätigkeiten = [ ] tätig keit en (active -> actions)
Tagung = [ ] tag ung ϕ (day -> meeting) ■

4.3.3 Adjectives

Adjectives can be declined with respect to five dimensions: gender, case, number, comparison, and substantival or pronominal use. They can only be preceded by simple prefixes:

ADJ = PREFIX <u>A-STEM</u> ENDING (4.5)

Example 4.3:

indirektesten = in direkt esten (most indirect) ■

In analogy to nouns also adjectives can be derived from verbs, substantives or other adjectives, these derived adjectives again can be formed by means of complex prefix lists:

ADJ = [PREFIX] <u>V-STEM | N-STEM | A-STEM</u> <u>SUFFIX</u> ENDING (4.6)

Example 4.4:

voraussichtliches = [vor, aus] sicht lich es (view -> presumable)
unlösbaren = [un] lös bar en (to solve -> unsolvable)
langsam = [] lang sam ϕ (long -> slow) ■

4.4.4 Verbs

Verbs are conjugated according to mood, number, person, and tense. They can only be accompanied by simple prefixes:

VERB = PREFIX <u>V-STEM</u> ENDING (4.7)

Most of the prefixes are separable from the word stem, they can occupy distant positions in an input sentence, a phenomenon which is dealt with by means of auxiliary dictionary entries for these prefixes.

Example 4.5:

durchführen = durch führ en (to accomplish)
Er *führte* die Lieferung termingerecht *durch*.
(He accomplished the delivery in time.) ■

The numerous irregular verb forms are modelled by storing them as different dictionary entries and by partitioning the corresponding conjugations. Derived verbs from nouns or adjectives were not mapped as morphological rules but are also realised by use of own entries because of their inherent irregularity. The way we felt about it, this was not a severe drawback. It is often only a matter of design whether a verb is regarded as derived from a noun or vice versa if one does not want to get lost in profound etymological details which have no relevant impact on the practical use. The final derivation structure is shown in Figure 8.

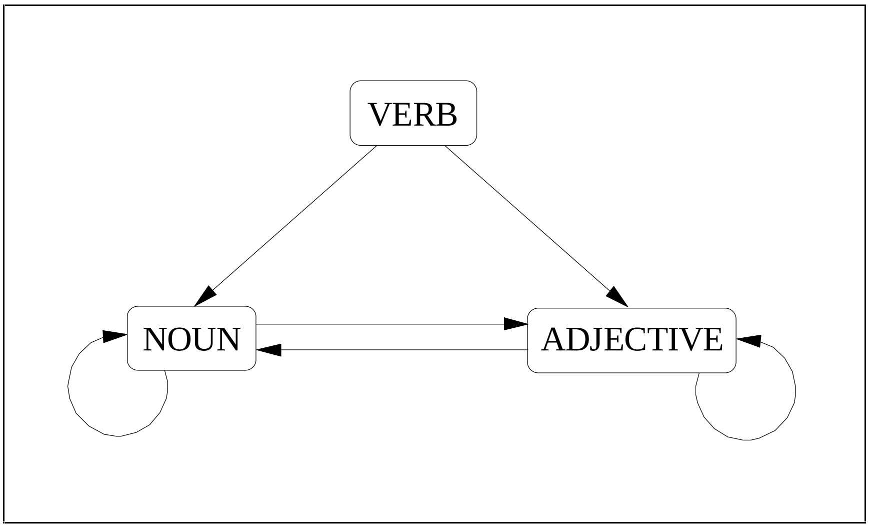

Figure 8: Derivation structure of complex word categories

An area of rich and tricky morphological phenomena constitutes the building of past participles including also adjectival uses. The following patterns cover all possible situations. The first one represents the normal case where the past participle is formed by use of the prefix *ge-* (PAP-PREFIX), the second one covers conditions where this prefix has to be replaced by another present prefix (REPL-PREFIX, prefixes capable of such substitutions are for example *ver-*, *ent-*, *er-*). Finally, in the third case the situation is figured that a verb takes no prefix at all (e.g. verbs derived from nouns by the suffix -*ieren*):

VERB = PREFIX PP-PREFIX V-STEM PAP-ENDING A-ENDING (4.8)

VERB = REPL-PREFIX V-STEM PAP-ENDING A-ENDING (4.9)

VERB = PREFIX V-STEM PAP-ENDING A-ENDING (4.10)

Example 4.6:

abgearbeitete = ab ge arbeit et e (to work -> worked off)

verkauft = ver kauf t ϕ (to buy -> sold)

aktualisierten = ϕ aktualisier t en (to update -> updated) ■

Finally, the formation of the present participle is comparatively simple and is determined by one single uniform pattern which again includes adjectival use:

VERB = PREFIX V-STEM PRP-ENDING A-ENDING (4.11)

Example 4.7:

laufenden = ϕ lauf end en (to run -> current) ■

4.4 Two-Level Rules

We applied the *two-level formalism* introduced by Koskenniemi [Koskenniemi83] to the correct treatment of three special German morphological phenomena, i.e. *ablaut*, *elision*, and *binding sounds*. Instead of adopting the realisation of the two-level rules by means of *finite state techniques*, we used them only as expressive framework for the precise specification of the morphological features which was translated into a corresponding set of deductive database rules [Winiwarter93b].

Although we could also have dealt with the morphological phenomena in question without the two-level formalism, we gained a significant increase of transparency and conciseness by means of its application. The most convincing evidence of this assertion was the reduction of required additional columns by the use of *archiphonemes* in the dictionary.

4.4.1 Ablaut

The ablaut is a special morphological feature of the German language concerning the transformation of the vowels *a, o, u* to *ä, ö, ü* in the stems of nouns, adjectives, and verbs in the course of deriving the following inflections:

- plurals of nouns
- comparatives and superlatives of adjectives
- second and third person singular of verbs

The presence or absence of the ablaut in the above cases is not subject of general systematic regularities (with very few exceptions, e.g. substantives which build the plural on *-er* always form the ablaut), so that it is marked lexically in the dictionary.

Example 4.8:

Wolf-Wölfe (wolf-wolves) vs. Stoff-Stoffe (material-materials)

klug-klüger (intelligent-more intelligent) vs. krumm-krummer (curved-more curved)

tragen-du trägst (to carry-you carry) vs. fragen-du fragst (to ask-you ask) ■

By making use of the two-level formalism, the condition for the ablaut (for the vowel a) can be formalised as follows [Karttunen87]:

A:ä ⇔ _=* +: %: (4.12)

The symbols used in this formula have to be interpreted as:

A .... *archiphoneme* which marks the existence as well as the position of the vowel for the ablaut at the lexical level (default assignment of surface representation: *a*)
: .... characterises corresponding pair of lexical and surface representation
ä .... transformed surface character if condition is satisfied
⇔ ...... condition for obligatory transformation (if condition is true, the transformation must be performed)
_ .... position of archiphoneme in the dictionary entry

= .... arbitrary character
* .... 0 or any number of repetitions
+: .... morpheme boundary
%: ...... test for syntactic feature (e.g. plural of noun)

The rule can therefore be verbalised in the following way: the archiphoneme *A* is transformed into *ä* at the surface level if it is followed by a sequence of arbitrary characters and the morpheme boundary, in addition to that the test for the specified syntactic feature must be satisfied. In all other cases it becomes the default value *a*. So the above mentioned examples can now be easily distinguished (e.g. klUg vs. krumm) in the dictionary. The syntactic test which has to be applied is selected correctly in accordance with the actual word category [Emele88].

In addition to the three above mentioned cases of the presence of the ablaut there also exists the possibility that a vowel-gradation may occur in the course of the process of word derivation, e.g. *Tag->täglich (day->daily)*. This modification is not subject to any regularities either but depends only on the specific combination. As it comprises only a backward reference to the position to be altered, it could not be resolved by the use of an archiphoneme, but we added an appropriate characteristic in the dictionary instead.

4.4.2 Elision

Elision is the omission of the unstressed e-sound. It can be distinguished on the one hand between obligatory and optional omissions, on the other hand between elisions concerning the stem or the ending. While elisions occurring in inflexional endings can easily be handled in the dictionary by corresponding morphological features, the former case results in the need for a concise representation by means of a two-level rule, avoiding redundant lexical entries.

In contrast to the processing of the ablaut, there exist three generally valid rules for the presence of elision. Therefore, they can be taken into account by the logical rules of the deductive database.

➲ If the stem of an adjective ends in *-el* and an ending starting with *-e* is appended, then *-el* has to be reduced to *-l*:

e: 0 ⇔ _ l +: e (4.13)

Example 4.9:

variabel en -> variablen (variable) ■

➲ If the stem of a verb ends in *-el* and an ending starting with *e* is appended, then *-el* can be reduced to *-l* (optional reduction is expressed in the formula by ⇒ instead of ⇔):

e: 0 ⇒ _l +: e (4.14)

Example 4.10:

handel e -> handele, handle (to act) ■

➲ If the stem of an adjective or a verb ends in *-er* and an ending starting with *e* is appended, then *-er* can be reduced to *-r*:

$$e{:}\ 0 \ \Rightarrow \_\ r\ +{:}\ e \tag{4.15}$$

Example 4.11:

änder e -> ändre, ändere	(to change)	
finster e -> finstre, finstere	(dark)	■

In addition to these universal rules there is also the possibility that word derivations might lead to elisions (required or optional) if the stem ends in *-el* or *-er* and the suffix starts with a vowel, e.g. *handeln->Handlung (to act->action)*. This case is modelled in analogy with the ablaut occurring in derivations.

4.4.3 Binding Sounds

One final morphological characteristic of the German language is the insertion of so-called binding sounds (*s*, *e* or *n*) which tie together the individual parts of a word in the formation of derived or compound words. Again, like in the case of the ablaut, there exist no universal rules whether a specific compound is formed with or without a binding sound, so the problem is once more solved directly by the use of archiphonemes in the dictionary (stated here for the binding sound s):

$$S{:}\ s \Leftrightarrow \_\ +{:}\ = =^{*}\ +{:} \tag{4.16}$$

The archiphoneme *S* is represented as *s* at the surface level if it is directly followed by the morpheme boundary and a sequence of arbitrary characters (at least one), constituting the appended morpheme.

Example 4.12:

[ein, kaufS] preis -> Einkaufspreis	(cost price)	■

4.5 Implementation

4.5.1 Database Schema

In the following we give some representative examples of applied base predicates for the categories stated by our morpho-syntax. Only syntactic properties are considered, semantic features will be treated in *Chapter 6. Semantic and Pragmatic Analysis*.

✍ *Particles:*

konjunktion(	category: conjunction
Stamm: string,	stem
Hierarchie: string,	hierarchy: co-ordination, sub-ordination
Bindungsart: string).	binding category: copulative, disjunctive etc.

Nouns:

substantiv(	category: noun
Stamm: string,	stem
Geschlecht: string,	gender
Deklinationsklasse: integer,	declination class
Praefixe: {[string]},	prefix sequences
Suffixe: {(string, integer,	suffixes for derivations, ablaut type, elision type,
integer, {[string]})}).	lists of required prefix sequences
deklination(	
Deklinationsklasse: integer,	declination class
Endungen: {(string, {(string, string)})}).	endings, sets of syntactic features: (case, number)
substsuffix(	suffix for deriving nouns
Stamm: string,	stem
Geschlecht: string,	gender
Deklinationsklasse: integer).	declination class

Adjectives:

adjektiv(	category: adjective
Stamm: string,	stem
Deklinationsklasse: integer,	declination class
Praefixe: {string},	prefixes
Suffixe: {(string, integer,	suffixes for derivations, ablaut type, elision type,
integer, {[string]})}).	lists of required prefix sequences
adjdekl(	
Deklinationsklasse: integer,	declination class
Endungen: {(string,	endings, sets of syntactic features:
{(string, string, string, string, string)})}).	(comparison, substantival or pronominal use, number, gender, case)
adjsuffix(	suffix for deriving adjectives
Stamm: string,	stem
Deklinationsklasse: integer).	declination class

Verbs:

verb(	category: verb
Stamm: string,	stem
Subjekttyp: string,	impersonal or personal verb
Objekttyp: string,	transitive or intransitive verb
Reflexivitaet: string,	reflexive or irreflexive verb
Verwendungsart: string,	full verb, auxiliary verb, modal verb
Konjugationsklasse: integer,	conjugation class
Partizipklasse: integer,	past participle class
Praefixe: {(string, string, string)},	prefixes, new transitivity, new reflexivity
Suffixe: {(string, integer,	suffixes for derivations, ablaut type, elision type,
integer, {[string]})}).	lists of required prefix sequences

konjugation(	
Konjugationsklasse: string,	conjugation class
Endungen: {(string,	endings, sets of syntactic features:
{(string, integer, string, string)})}).	(mood, person, number, tense)

4.5.2 Facts

The individual dictionary entries are inserted into the facts file, in other words they constitute the real data or population of the database (tuples). Please notice the elimination of capitalisation and the internal representation of the special characters *ä, ö, ü, ß* by their international transcriptions *ae, oe, ue, ss*. Another interesting detail is the use of the archiphonemes defined in the previous chapter and the coding of ablauts (1 ... present, 0 ... absent) and elisions (2 ... required, 1... optional, 0 ... absent) for derivations as the following sample entries illustrate:

✍ *Particles:*

konjunktion(	category: conjunction
und,	stem [=and]
koordinierend,	co-ordination
kopulativ).	copulative

✍ *Nouns:*

substantiv(	
preis,	stem [=price]
masculinum,	gender
6,	declination class
{[ein,kaufS]},	prefix sequence yielding *Einkaufspreis* [=cost price]
{(lich,0,0,{[ ]})}).	suffix yielding adjective *preislich* [=estimable] (neither ablaut nor elision present)

deklination(	
6,	declination class
{(´´, {(nominativ, singular),	sets of endings, syntactic features: (case, number)
(dativ, singular),	
(akkusativ, singular)}),	
...}).	
substsuffix(	
e,	stem
femininum,	gender
1).	declination class
substsuffix(	
ung,	stem
femininum,	gender
3).	declination class

Adjectives:

adjektiv(	
lAng,	stem [=long]
1,	declination class
{},	prefixes
{(e, 1, 0, {[ ]}),	suffix yielding *Länge* [= length]
(lich, 1, 0, {[ ]})}).	suffix yielding *länglich* [=longish]

adjdekl(	
1,	declination class
{(es, {(positiv, pronominal,	sets of endings, syntactic features:
nominativ, singular, neutrum)}),	(comparison, substantival or pronominal use,
...}).	number, gender, case)

adjsuffix(	
lich,	stem
1).	declination class

Verbs:

verb(	
zoeger,	stem [=to hesitate]
persoenlich,	personal verb
intransitiv,	intransitive verb
irreflexiv,	irreflexive verb
hauptzeitwort,	full verb
1,	conjugation class
1,	past participle class (prefix *ge*, ending *t*)
{(ver, transitiv, reflexiv)},	prefix yielding *verzögern* [=to delay] (transitive, reflexive)
{(ung, 0, 0, {[ver]})}).	suffix yielding noun *Verzögerung* [=delay] (neither ablaut nor elision present)

konjugation(	
1,	conjugation class
{(st, {(indikativ, 1, singular, praesens)}),	endings, sets of syntactic features:
...}).	(mood, person, number, tense)

4.5.3 Logical Rules

Since the morphological data is already specified in such a complex, yet also clear and compact way, the rule file of the deductive database is accordingly simple and straightforward. By use of derived predicates new conclusions are inferred from the base predicates representing the dictionary, resulting in a complete morphological analysis of the input sentence.

On the top-level the following predicate which handles the I/O-functions is defined:

```
ma <-                            top-level of morphological analysis
    input(Satz),                 external C-predicate for requesting sentences
                                 from user, separating the individual words and
                                 transforming them into a list of atoms
    suche(Satz, Ergebnis),       proper morphological analysis
    output(Ergebnis).            formatted output of analysis results
```

The next step comprises the analysis of each individual word. In order to reduce the processing time some special character patterns as well as abbreviations are tested by the following predicate (also including external C-predicates for string-processing) before the dictionary is accessed:

speztest(Wort, Wort2, Typ) (4.17)

Wort .... analysed word
Wort2 ..... expansion of abbreviations, otherwise it equals Wort
Typ .... type of special pattern

These patterns include the following important types:

- punctuation marks
- numbers
- date, time, and currency formats
- physical units

Only if none of these special patterns is detected, the analysis is continued by examining the dictionary, the resulting argument Typ of the pre-test predicate is then marked as unknown. The complete rule for recursively analysing the words of the input list takes the following format:

```
suche([Wort|Rest], [Ergebnis|Rest2]) <-
    speztest(Wort, Wort2, Typ),          pre-test for special patterns and abbreviations
    if(Typ~=unknown                      if word type is not unknown
    then                                 then
        Ergebnis={(Wort2, Typ,               result equals word (possibly expanded), type,
                ([ ], [ ], [ ], [ ]))}       empty morphological structure
    else                                 else
        if(stammtest(Wort2, Ergebnis2)       if word can be analysed correctly
        then                                 then
            Ergebnis=Ergebnis2                   result equals result of word analysis
        else                                 else
            Ergebnis={(Wort2, unknown,           word is marked as unknown
                ([ ], [ ], [ ], [ ]))})),
    suche(Rest, Rest2).                  analysis of next word

suche([ ],[ ]).                          exit rule of recursion
```

To include the correct representation of ambiguous analysis results as a set of possible interpretations, the various outcomes at the word level are comprised as elements of a single set by means of the grouping operator (<...>):

```
stammtest(Wort, <Ergebnis>) <-              grouping of results at word level into single set
        wort(Wort, Ergebnis).
```

The predicate wort realises the dictionary level of the analysing process and consists of different rules for each word category, e.g. for the simple category of conjugations:

```
wort(Eintrag, (Eintrag, konjunktion,        classifying word as conjunction
    ([ ], [ ], [ ], [ ]))) <-                   yielding as result: stem, word category,
                                                morphological structure
    konjunktion(Eintrag,_,_).               searching for conjunctions in dictionary
```

As one complex illustrative example we choose the derivation of nouns, the other implemented rules can be inferred from this example in an easy and straightforward way:

```
wort(Wort, (Eintrag3, ableitSubst,          classifying word as derived noun
    (Praefix2, Suffixstamm, Endung, [ ]))) <-   yielding as result: stem, word category,
                                                morphological structure
    verb(Eintrag,_,_,_,_,_,_,_, Suffixe),   searching for verbs in dictionary
    ablaut(Eintrag, Eintrag2, Ablauttyp),   possible modifications are generated as additional
                                                solutions (external C-predicate)
    elision(Eintrag2,Eintrag3,Elisionstyp), as above
    affixe(Wort, Eintrag3, Praefix, Suffix), sub-string test for verb stem (external C-predicate)
                                                if satisfied it yields the separated prefix and
                                                suffix part
    suffixtest(Suffix, Suffixe, Ablauttyp,  checking suffix with suffixes in dictionary resulting
      Elisionstyp, Suffixstamm,                 in suffix stem and ending
      Endung, Praefixe),
    praefixtest(Praefix, Praefixe, Praefix2). checking prefix with possible prefix sequences

suffixtest(Suffix, Suffixe, Ablauttyp,      suffix test
      Elisionstyp, Suffixstamm,
      Endung, Praefixe) <-
    member((Suffixstamm, Ablauttyp,         checking each possible suffix in set (verifying also
       Elisionstyp, Praefixe), Suffixe),        accordance with regard to ablaut and elision)
    affixe(Suffix, Suffixstamm,´´, Endung), separating ending from suffix stem
    substsuffix(Suffixstamm,_, Endungen),   retrieving possible endings
    member((Endung,_), Endungen).           checking ending with set of valid endings

praefixtest(Praefix, Praefixe, Praefix2) <- prefix test rule for case where prefixes are present
    Praefix~=´´,                            rule only applies if prefixes are present
    member(Praefixliste, Praefixe),         retrieving possible prefix sequences
    listtest(Praefix, Praefixliste, Praefix2). recursive checking of prefix with valid
                                                prefix sequences

praefixtest(´´, Praefixe, [ ]) <-           prefix test rule for case where no prefix is present
    member([ ], Praefixe).
```

listtest(Praefix, [Praefix2|Rest], [Praefix3 | Rest2]) <-	recursive checking of prefix with valid prefix sequences
Rest~=[ ],	rule only applies if more than one prefix present
binding(Praefix2, Praefix3),	external C-predicate for transforming binding sound, otherwise *Praefix3* equals *Praefix2*
affixe(Praefix, Praefix3, ´´, Praefrest),	separating first prefix and checking it against list entry
listtest(Praefrest, Rest, Rest2).	analysis of the next prefix in the list
listtest(Praefix, [Praefix2], [Praefix]) <-	exit rule for single prefix
binding(Praefix2, Praefix).	external C-predicate for transforming binding sound, otherwise *Praefix* equals *Praefix2*

4.5.4 Query Forms

Since the I/O-functions are handled by external C-predicates and special I/O-predicates of LDL, there is no need for defining any complicated query forms by declaring covered and free variables. The only thing that has to be done in the query forms file is to specify the main predicate ma by means of the statement: qform ma. After the processing of all definitions neatly separated into the four file types, the program is simply started by typing ?ma leading to the invitation to the user to enter an input sentence.

The complicated resulting morphological structure contained in the variable Ergebnis is of course only of use for internal purposes, that is, it constitutes the basis for the subsequent steps of analysis. By means of the predicate output we therefore produced an informative formatted test output including the precise lemmatisation, the individual morphemes as well as all interesting syntactic information.

Example 4.13:

As a short illustrative example we give the analysis result for the following sentence (analysis of numbers is only mentioned once):

Bestellungen 3 und 4 verzögern sich 5 Tage
[=Deliveries 3 and 4 are delayed by 5 days]
(literally: Deliveries 3 and 4 delay themselves 5 days)

Bestellungen	*deliveries*
Stamm: stell	stem
Wortart: abgeleitetes Substantiv	category: derived noun
Präfixliste: [be]	prefix sequence
Suffix: ung	suffix
Geschlecht: femininum	gender
Endung: en	ending
Deklination: {(nominativ, plural), (genitiv, plural), (dativ, plural), (akkusativ, plural)}	set of syntactic features: (case, number)
3	*3*
Stamm: 3	stem
Wortart: Ganzzahl	category: integer

und	*and*
Stamm: und	stem
Wortart: Konjunktion	category: conjunction
koordinierend	co-ordination
kopulativ	copulative
verzögern	*to delay*
Stamm: zoeger	stem
Wortart: Verb	category: verb
persoenlich	personal verb
transitiv	transitive verb
reflexiv	reflexive verb
Hauptzeitwort	full verb
Präfix: ver	prefix
Endung: n	ending
Konjugation: {(indikativ, 1, plural, praesens), (infinitiv, 0, ´´, ´´), (indikativ, 3, plural, praesens), (imperativ, 3, plural, ´´)}	set of syntactic features: (mood, person, number, tense)
sich	*himself, herself, itself, themselves*
Stamm: sich	stem
Wortart: Reflexivpronomen	category: reflexive pronoun
Deklination: {(3, singular, dativ), (3, plural, dativ), (3, singular, akkusativ), (3, plural, akkusativ)}	set of syntactic features: (person, number, case)
Tage	*days*
Stamm: tag	stem
Wortart: Substantiv	category: noun
Präfix: [ ]	prefix sequence
Geschlecht: masculinum	gender
Endung: e	ending
Deklination: {(nominativ, plural), (genitiv, plural), (akkusativ, plural)}	set of syntactic features: (case, number)

■

4.6 Summary

This Section dealt with morphological and lexical analysis, the first two steps of natural language analysis which cannot be regarded as separated within IDA interfaces. This is due to the selected lexical approach of storing only canonical forms as dictionary entries and attaching to them all morphological features, also considering complex cases of prefixes, derivations, and compound words.

After developing the required formal framework consisting of the morpho-syntax and two-level rules, the theoretical concepts were mapped to an efficient LDL implementation. By applying a hierarchical architecture, we achieved a compact dictionary as excellent basis for further syntactic and semantic analysis.

5. Syntactic Analysis

5.1 Introduction

The aim of syntactic analysis within natural language interfaces should not be the coverage of the complete language, especially not of all those exotic phenomena possessing only linguistic evidence but no practical relevance. Whilst the generality is therefore on the one hand restricted in comparison with other applications of natural language processing, it has to provide on the other hand important extensions indispensable for effective information retrieval [Bates87]:

- tolerance as concerns ungrammatical sentences
- correct interpretation of incomplete sentences
- processing of unknown words

The formal specification of the valid constructs of a language is defined by a *grammar* whereas the *parser* represents the tool to analyse a sentence, it verifies the compatibility of a sentence with the grammar. With regard to their expressive power the grammars can be classified in *context-free grammars, augmented grammars,* and *unification grammars* [Allen87].

Context-free grammars have their origin in the analysis of formal languages and were adapted to the processing of natural language [Nijholt88]. Therefore, many special problems of natural language could not be treated efficiently. This weakness led to the augmentation of syntactic features and local registers to the grammar symbols [Charniak86]. These *augmented grammars* were now able to keep track of the sentence structure and to check for the agreement of syntactic properties, e.g. case, number or person.

In order to obtain a more compact and natural representation the conditions and assignments used within augmented grammars were replaced by the operation of unification derived from logic programming. Two structures are unified in that all registers which exist only in one of the two structures are copied. If a register exists in both structures, the intersection is computed. The resulting *unification grammars* detect missing agreements already at an early point of analysis, leading the way to efficient parser implementations [Gazdar89].

A problem of special importance for languages with free word order like German is the *constituent transfer*, i.e. situations in that some phrases are moved from its expected positions in the sentence [Winograd83]. Examples are inversions in questions or dependent clauses, the separation of constituent parts by embedded phrases, and the topicalisation of constituents. Many techniques to deal with this phenomenon have been proposed but most of them lack of declarative expressiveness.

Section 5.2 Grammar Formalisms gives a view over some of the most influential grammar formalisms for natural language processing also providing examples of handling constituent transfer. For a good survey see also [Allen87].

Based on this survey we select *Categorial Unification Grammar* as theoretical framework for syntactic analysis and introduce in *Section 5.3 Extended Categorial Unification Grammar* six important new extensions in order to gain expressive power for the processing of free word order languages. Finally, we show in *Section 5.4 Implementation* how a parser for our proposed grammar can be implemented efficiently in IDA.

5.2 Grammar Formalisms

5.2.1 Phrase Structure Grammar

A context-free phrase structure grammar [Chomsky56] consists of a set of *derivation rules*:

$$\text{Symbol} \leftarrow \text{Symbol1} \quad \text{Symbol2} \quad \ldots \tag{5.1}$$

The symbols represent the *constituents* of the sentence. It can be distinguished between *non-terminals* which can be transformed by the application of rules and *terminals* which correspond to the word categories stored in the dictionary.

Example 5.1:

The following simple grammar accepts sentences which consist of a noun phrase and a verb phrase, the latter is formed out of a verb with an optional noun phrase and an optional prepositional phrase.

1. S	←	NP VP
2. NP	←	ART NOUN
3. NP	←	NAME
4. PP	←	PREP NP
5. VP	←	VERB
6. VP	←	VERB NP
7. VP	←	VERB NP PP
8. VP	←	VERB PP

■

With regard to the applicability of parsing algorithms, two basic techniques exist: *top-down parsing* and *bottom-up parsing* [Aho72].

Example 5.2:

The two basic parsing techniques are demonstrated by use of the analysis of the example sentence *Hugo aß das Eis (=Hugo ate the ice-cream)* and the grammar from *Example 5.1*, the numbers indicate the application of the corresponding rules, no numbering represents a simple replacement by means of the dictionary.

Top-down parsing			Bottom-up parsing		
		S		→	Hugo aß das Eis
1.	→	NP VP		→	NAME aß das Eis
3.	→	NAME VP		→	NAME VERB das Eis
	→	Hugo VP		→	NAME VERB ART Eis
6.	→	Hugo VERB NP		→	NAME VERB ART NOUN
	→	Hugo aß NP	3.	→	NP VERB ART NOUN
2.	→	Hugo aß ART NOUN	2.	→	NP VERB NP
	→	Hugo aß das NOUN	6.	→	NP VP
	→	Hugo aß das Eis	1.	→	S

■

These two parsing methods possess as they are striking disadvantages:

- If a top-down parser rejects a grammar rule, all derivations for non-terminals are deleted. Therefore, if the same symbol appears later within another rule, the derivation must be calculated once again as for example the symbol NP in:

  VP ← VERB NP
  VP ← VERB NP PP

- A bottom-up parser generates many superfluous non-terminals which can never take this position in the sentence, e.g. in the following sentence the non-terminal VP is derived before the parser realise that the subject is missing:

  Aß das Eis mit dem Löffel (Ate the ice-cream with the spoon)

Therefore, most practical applications use *mixed-mode parsers*. Classical examples of efficient parsing algorithms are *chart-based parsers* [Kay73] and *Earley´s parser* [Earley70], for a good survey of recent approaches see [Tomita91].

In order to be able to check syntactic agreements and to record the sentence structure, *features* are added to the dictionary entries and *local registers* are attached to the non-terminals. If additionally *unification operations* are introduced, one results in *phrase structure unification grammars* [Shieber86].

Example 5.3:

For the four dictionary entries in *Example 5.2* the feature for the number singular is added as set, e.g.:

Hugo (NAME NUM {SING})

Each non-terminal is supported by a local register in order to test the agreement of number and to construct the syntactic structure:

(NP	ART	(PP	PREP	(VP	VERB	(S	SUBJ
	HEAD		NP)		OBJ		PRED
	NAME				POBJ		NUM)
	NUM)				NUM)		

Finally, the grammar in *Example 5.1* is augmented by unification operations:

1. S	←	NP VP	$NUM=NUM_{NP}=NUM_{VP}$, SUBJ=NP, PRED=VP
2. NP	←	ART NOUN	$NUM=NUM_{ART}=NUM_{NOUN}$, HEAD=NOUN, ART=ART
3. NP	←	NAME	$NUM=NUM_{NAME}$, NAME=NAME
4. PP	←	PREP NP	PREP=PREP, OBJ=NP
5. VP	←	VERB	$NUM=NUM_{VERB}$, VERB=VERB
6. VP	←	VERB NP	$NUM=NUM_{VERB}$, VERB=VERB, OBJ=NP
7. VP	←	VERB NP PP	$NUM=NUM_{VERB}$, VERB=VERB, OBJ=NP, POBJ=PP
8. VP	←	VERB PP	$NUM=NUM_{VERB}$, VERB=VERB, POBJ=PP

If the example sentence

Hugo aß das Eis mit dem Löffel (Hugo ate the ice-cream with the spoon)

is analysed, this results in the following sentence structure:

```
(S   SUBJ  (NP  NAME  Hugo
                NUM   {SING})
     PRED  (VP  VERB  aß
                OBJ   (NP  ART   das
                           HEAD  Eis
                           NUM   {SING})
                POBJ  (PP  PREP  mit
                           OBJ   (NP  ART   dem
                                      HEAD  Löffel
                                      NUM   {SING}))
                NUM   {SING})
     NUM   {SING})
```
■

The constituent transfer is dealt with a so-called *hold list*. Every time an unexpected constituent is found, it is stored in the hold list by use of the *HOLD-operation* so that it can be retrieved later again by a corresponding *VIR-operation* [Allen87]. A sentence is only analysed successfully if the final hold list is empty.

Example 5.4:

The grammar in *Example 5.1* is extended by the possibility to analyse sentences where the prepositional object is topicalised, e.g.:

Mit dem Löffel aß Hugo das Eis (With the spoon ate Hugo the ice-cream)

1. S ← HOLD(PP) HOLD(V) S'
2. S ← S'
3. S' ← NP VP
4. NP ← ART NOUN
5. NP ← NAME
6. PP ← PREP NP
7. VP ← V
8. VP ← V NP
9. VP ← V NP PP
10. VP ← V PP
11. V ← VERB
12. V ← VIR(VERB)
13. PP ← VIR(PP) ■

An important derivative of phrase structure grammars is the *generalized phrase structure grammar (GPSG)* introduced by Gazdar [Gazdar82]. GPSG provides on the one hand very powerful notions in order to formalise even the trickiest syntactic problems of natural language. On the other hand, this complexity has as consequence that only simplified versions can be parsed efficiently (e.g. [Evans87, Fisher 89]).

Within GPSG the context-free grammar rules are split up in *ID (Immediate Dominance)* and *LP (Linear Precedence) rules* in that the ID rules contain only the information about the nature of the constituents whereas the LP rules define the sequence conditions.

Example 5.5:

The context-free rule from *Example 5.1*.

S ← NP VP

is represented in GPSG as follows:

ID: S ← NP, VP
LP: NP < VP ■

Instead of local registers the features are directly attached to the non-terminals resulting in complex symbols called *categories*, e.g. NP[SING]. For the correct processing of features two techniques are provided:

- a set of *conventions* or *constraints* that control the automatic propagation of features
- a set of *propositions* that are required to hold (so-called *feature co-occurrence restrictions*)

In GPSG constituent transfer can be dealt with by use of *slashed categories* indicating categories where a constituent is missing, e.g. S/NP. Other advanced techniques include *meta rules* and *variable categories* [Gazdar85]. For example, the following rule generates for each active verb phrase the corresponding passive one by removing the noun phrase and replacing it by an optional prepositional phrase, the variable category X stands for any sequence of constituents:

$$VP \leftarrow VERB[TR], NP, X \Rightarrow VP[PAS] \leftarrow VERB, X, (PP[by]) \quad (5.2)$$

5.2.2 Transition Nets

In context-free *recursive transition nets* [Woods70] the grammar rules are mapped to directed graphs, each sub-graph representing a non-terminal. The analysis of a sentence corresponds to the traversal of the graph, it is successful if the traversal ends in a valid final state.

The following arc labels exist:

- CAT is satisfied if the actual word belongs to the required category
- PUSH calls sub-graph, satisfied if traversal successful
- JUMP is always satisfied
- POP returns after successful traversal to prior sub-graph

Example 5.6:

Figure 9 shows the corresponding grammar to *Example 5.1*. The example sentence from *Example 5.2* is parsed in that for each non-terminal the sentence position, the calling states, and the selected arc is indicated. The arcs are numbered downward. If more then one arc can be applied, the top arc is selected, the other arcs are stored for backtracking.

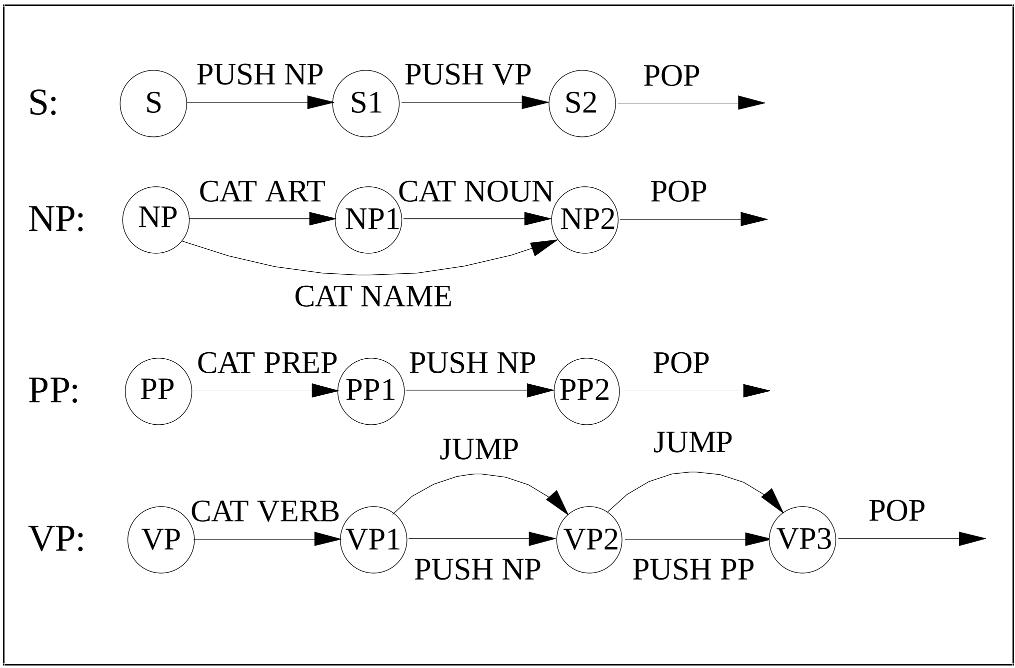

Figure 9: Example of recursive transition network

$_1$Hugo$_2$aß$_3$das$_4$Eis$_5$

Actual state	Arc	Backtracking states
1. (S, 1, NIL)	S/1	NIL
2. (NP, 1, (S1))	NP/2	NIL
3. (NP2, 2, (S1))	NP2/1	NIL
4. (S1, 2, NIL))	S1/1	NIL
5. (VP, 2, (S2))	VP/1	NIL
6. (VP1, 3, (S2))	VP1/1	NIL
7. (VP2, 3, (S2))	VP2/1	(NP, 3, (VP2, S2))
8. (VP3, 3, (S2))	VP3/1	(NP, 3, (VP2, S2)), (PP, 3, (VP3, S2))
9. (S2, 3, NIL)	S2/1	(NP, 3, (VP2, S2)), (PP, 3, (VP3, S2))
10. (NP, 3, (VP2, S2))	NP/1	(PP, 3, (VP3, S2))
11. (NP1, 4, (VP2, S2))	NP1/1	(PP, 3, (VP3, S2))
12. (NP2, 5, (VP2, S2))	NP2/1	(PP, 3, (VP3, S2))
13. (VP2, 5, (S2))	VP2/1	(PP, 3, (VP3, S2))
14. (VP3, 5, (S2))	VP3/1	(PP, 3, (VP3, S2)), (PP, 5, (VP3, S2))
15. (S2, 5, NIL)	S2/1	(PP, 3, (VP3, S2)), (PP, 5, (VP3, S2)) ■

Similar to phrase structure grammars, in *Augmented Transition Networks (ATN)* the directed graphs are augmented by features, local registers, conditions, and assignments [Woods73, Kaplan73, Bates78]. The asterisk symbol is used in this context to indicate the features of the word for CAT arcs and the local register of the sub structure for PUSH arcs. Unification operations do not exist, they are approximated by use of *instantiation rules* [Winograd83] which make it possible to pass arguments to PUSH arcs so that agreement conditions can already be tested in the called sub-graph.

Example 5.7:

The following annotations extend the network in Figure 5 resulting in the same grammar as in *Example 5.3*. Additionally, the instantiation rule NUM ← NUM for state S1/1 is included to check the agreement of the number of subject and predicate.

Arc	Conditions	Assignments
S/1		SUBJ ← *, NUM ← NUM*
S1/1		PRED ← *, NUM ← NUM*
NP/1		NUM ← NUM*, ART ← *
NP1/1	NUM∩NUM*	NUM ← NUM∩NUM*, HEAD ← *
NP/2		NUM ← NUM*, NAME ← *
PP/1		PREP ← *
PP1/1		OBJ ← *
VP/1	NUM∩NUM*	NUM ← NUM∩NUM*, VERB ← *
VP1/2		OBJ ← *
VP2/2		POBJ ← *

■

Constituent transfer is again handled by use of a hold list. The *HOLD operation* is added to the assignment part whereas the *VIR operation* is modelled as arc label [Nijholt88].

Example 5.8:

Figure 10 shows an ATN which, together with the following assignments, is able to analyse the same sentences as in *Example 5.4*.

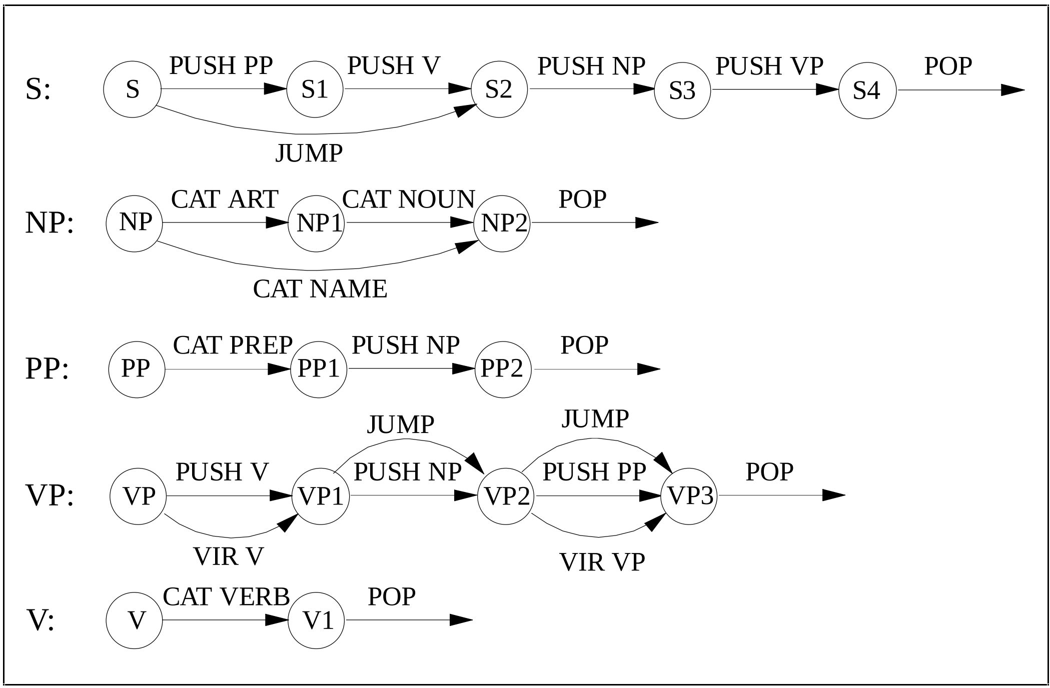

Figure 10: Example of ATN dealing with constituent transfer

Arc	Assignments
S/1	HOLD*
S1/1	HOLD*
S2/1	SUBJ ← *
S3/1	PRED ← *
NP/1	ART ← *
NP1/1	HEAD ← *
NP/2	NAME ← *
PP/1	PREP ← *
PP1/1	OBJ ← *
VP/1	VERB ← *
VP/2	VERB ← *
VP1/2	OBJ ← *
VP2/2	POBJ ← *
VP2/3	POBJ ← *

■

5.2.3 Logic Grammar

In *logic grammars* the context-free rules are written as *horn clauses* which are proven in a top-down manner [Colmerauer78, Pereira80]. A complete proof corresponds to the successful analysis of a sentence. As additional arguments to the non-terminals the starting and ending position in the sentence are taken.

Example 5.9:

The following logic grammar is equivalent to the phrase structure grammar in *Example 5.1*, it is written in LDL notation.

```
s(P1, P3)       ← np(P1, P2), vp(P2, P3).
np(P1, P3)      ← art(P1, P2), noun(P2, P3).
np(P1, P2)      ← name(P1, P2).
pp(P1, P3)      ← prep(P1, P2), np(P2, P3).
vp(P1, P2)      ← verb(P1, P2).
vp(P1, P3)      ← verb(P1, P2), np(P2, P3).
vp(P1, P4)      ← verb(P1, P2), np(P2, P3), pp(P3, P4).
vp(P1, P3)      ← verb(P1, P2), pp(P2, P3).
```

For each terminal a horn clause has to be appended which consists of the predicate word (reading the word from the sentence) and a test for the word category in question.

```
verb(P1, P2)    ← word(Word, P1, P2), isverb(Word).
art(P1, P2)     ← word(Word, P1, P2), isart(Word).
noun(P1, P2)    ← word(Word, P1, P2), isnoun(Word).
name(P1, P2)    ← word(Word, P1, P2), isname(Word).
prep(P1, P2)    ← word(Word, P1, P2), isprep(Word).
```

The individual words of the analysed sentence are then mapped to the predicate word:

$_1$Hugo$_2$aß$_3$das$_4$Eis$_5$

```
word('Hugo', 1, 2).
word(aß, 2, 3).
word(das, 3, 4).
word('Eis', 4, 5).
```

Finally, the categories of the words have to appended as predicates:

```
isname('Hugo').
isverb(aß).
isart(das).
isnoun('Eis').
```

Now the sentence can be parsed by keeping track of backtracking states:

Actual state	Backtracking states
1. s(1, 5)	
2. np(1, P2), vp(P2, 5)	
3. art(1, P3), noun(P3, P2), vp(P2, 5)	name(1, P2), vp(P2, 5)
4. name(1, P2), vp(P2, 5)	
5. vp(2, 5)	
6. verb(2, 5)	verb(2, P2), np(P2, 5) verb(2, P2), np(P2, P3), pp(P3, 5) verb(2, P2), pp(P2, 5)
7. verb(2, P2), np(P2, 5)	verb(2, P2), np(P2, P3), pp(P3, 5) verb(2, P2), pp(P2, 5)
8. np(3, 5)	verb(2, P2), np(P2, P3), pp(P3, 5) verb(2, P2), pp(P2, 5)
9. art(3, P2), noun(P2, 5)	name(3, 5) verb(2, P2), np(P2, P3), pp(P3, 5) verb(2, P2), pp(P2, 5)
10. noun(4, 5)	name(3, 5) verb(2, P2), np(P2, P3), pp(P3, 5) verb(2, P2), pp(P2, 5)
11. ()	name(3, 5) verb(2, P2), np(P2, P3), pp(P3, 5) verb(2, P2), pp(P2, 5)

■

By augmenting additional arguments, agreement conditions can be checked and the sentence structure can be recorded. Of course, unification is realised very easily because if two arguments use the same symbol, the two are unified automatically [Shieber84].

Example 5.10:

The following grammar is again equivalent to the one in *Example 5.3.* The feature of the number is stored in the argument Num.

```
s(P1, Num, s(Subj, Pred), P3) ←
                              np(P1, Num, Subj, P2),
                              vp(P2, Num, Pred, P3).
np(P1, Num, np(Art, Head), P3) ←
                              art(P1, Num, Art, P2),
                              noun(P2, Num, Head, P3).
np(P1, Num, np(Name), P2) ←
                              name(P1, Num, Name, P2).
pp(P1, pp(Prep, Obj), P3) ←
                              prep(P1, Prep, P2),
                              np(P2, Num, Obj, P3).
vp(P1, Num, vp(Verb), P2) ←
                              verb(P1, Num, Verb, P2).
vp(P1, Num, vp(Verb, Obj), P3) ←
                              verb(P1, Num, Verb, P2),
                              np(P2, Num1, Obj, P3).
vp(P1, Num, vp(Verb, Obj, Pobj), P4) ←
                              verb(P1, Num, Verb, P2),
                              np(P2, Num1, Obj, P3),
                              pp(P3, Pobj, P4).
vp(P1, Num, vp(Verb, Pobj), P3) ←
                              verb(P1, Num, Verb, P2),
                              pp(P2, Num1, Pobj, P3).
```
■

The problem of constituent transfer is again resolved by applying a hold list mechanism [Pereira81] as shown in *Example 5.11*.

Example 5.11:

This grammar corresponds to that in *Example 5.4*. The first rule performs the HOLD operation whereas the last two rules are responsible for the VIR operation.

```
s(P1, P4, Hi, Ho) ← pp(P1, P2), verb(P2, P3),
                              s2(P3, P4, hold(verb, hold(pp, Hi)), Ho).
s(P1, P2, Hi, Ho) ← s2(P1, P2, Hi, Ho).
pp(P1, P2) ← prep(P1, P2), np(P2, P3).
s2(P1, P3, Hi, Ho) ← np(P1, P2), vp(P2, P3, Hi, Ho).
np(P1, P3) ← art(P1, P2), noun(P2, P3).
np(P1, P2) ← name(P1, P2).
vp(P1, P2, Hi, Ho) ← verb(P1, P2, Hi, Ho).
vp(P1, P3, Hi, Ho) ← verb(P1, P2, Hi, Ho), np(P2, P3).
vp(P1, P4, Hi, Ho) ← verb(P1, P2, Hi, H2), np(P2, P3), pp(P3, P4, H2, Ho).
vp(P1, P3, Hi, Ho) ← verb(P1, P2, Hi, H2), pp(P2, P3, H2, Ho).
verb(P1, P2, H, H) ← word(Word, P1, P2), isverb(Word).
pp(P1, P3, H, H) ← prep(P1, P2), np(P2, P3).
verb(P1, P1, hold(verb, H), H).
pp(P1, P1, hold(pp, H), H).
```
■

5.2.4 Lexical Functional Grammar

Lexical functional grammar (LFG) was introduced by Kaplan and Bresnan [Kaplan82] (see also [Berwick84]). It is a unification-based grammar in which the context-free phrase structure grammars are augmented by *functional annotations* representing the unification operations. By help of the phrase structure rules a *constituent structure (c-structure)* is constructed. To this c-structure equations are added which are derived from the functional annotations. The minimal solution of this system of equations (called *functional description)* forms the *functional structure (f-structure)*. The words contained in the dictionary are directly extended by functional annotations to indicate which features they have to assign to the local registers.

Example 5.12:

The following LFG is analogous to the grammar in *Example 5.3*. The local registers are indicated by ↑ for the calling structure and ↓ for the sub-structure, e.g. (↑SUBJ)=↓ in the first rule means that SUBJ of S is unified with the complete noun phrase.

S →	NP (↑SUBJ)=↓ (↑NUM)=(↓NUM)	VP ↑=↓	
NP →	ART	NOUN	
NP →	NAME		
PP →	PREP	NP (↑OBJ)=↓	
VP →	VERB		
VP →	VERB	NP (↑OBJ)=↓	
VP →	VERB	PP (↑POBJ)=↓	
VP →	VERB	NP (↑OBJ)=↓	PP (↑POBJ)=↓

In order to parse the example sentence from *Example 5.2* the following words have to be appended to the dictionary:

Hugo	NAME	(↑NAME) = 'Hugo', (↑NUM) = {SING}
aß	VERB	(↑VERB) = 'aß', (↑NUM) = {SING}
das	ART	(↑ART) = 'das', (↑NUM) = {SING}
aß	NOUN	(↑HEAD) = 'Eis', (↑NUM) = {SING}

The parsing of the sentence results in this functional description and c-structure:

S → NP VP
    (x1 SUBJ) = x2, (x1 NUM) = (x2 NUM), x1 = x3
NP → NAME
    (x2 NAME) = 'Hugo', (x2 NUM) = {SING}
VP → VERB NP
    (x3 VERB) = 'aß', (x3 NUM) = {SING}, (x3 OBJ) = x4
NP → ART NOUN
    (x4 ART) = 'das', (x4 HEAD) = 'Eis', (x4 NUM) = {SING}

The final solution of this system of equations gives the f-structure of the sentence:

$$x1 = x3 = \begin{bmatrix} SUBJ = x2 \\ NUM = \{SING\} \\ VERB = '\text{aß}' \\ OBJ = x4 \end{bmatrix}$$

$$x2 = \begin{bmatrix} NAME = '\text{Hugo}' \\ NUM = \{SING\} \end{bmatrix}$$

$$x4 = \begin{bmatrix} ART = '\text{das}' \\ HEAD = '\text{Eis}' \\ NUM = \{SING\} \end{bmatrix}$$

■

In LFG the constituent transfer is resolved by use of additional functional annotations (see [Winograd83]). The HOLD-operator is written as $\Downarrow$, the VIR-operator as $\Uparrow$.

Example 5.13:

The following grammar analyses the sentence from *Example 5.4.* The annotation $\downarrow=\Downarrow_{PP}$ indicates that a prepositional phrase is stored on the hold list whereas the last rule tries to replace a missing prepositional phrase from the hold list.

S →	PP $\downarrow=\Downarrow_{PP}$	V $\downarrow=\Downarrow_{V}$	S' $\uparrow=\downarrow$
S →	S' $\uparrow=\downarrow$		
S' →	NP $(\uparrow SUBJ)=\downarrow$	VP $\uparrow=\downarrow$	
NP →	ART	NOUN	
NP →	NAME		
PP →	PREP	NP $(\uparrow OBJ)=\downarrow$	
VP →	V $\uparrow=\downarrow$		
VP →	V $\uparrow=\downarrow$	NP $(\uparrow OBJ)=\downarrow$	
VP →	V $\uparrow=\downarrow$	PP $(\uparrow POBJ)=\downarrow$	
VP →	V $\uparrow=\downarrow$	NP $(\uparrow OBJ)=\downarrow$	PP $(\uparrow POBJ)=\downarrow$
V →	VERB		
V →	0 $\uparrow=\Uparrow_{V}$		
PP →	0 $\uparrow=\Uparrow_{PP}$		

With regard to the functional description one has to consider that the HOLD-operation assigns free variables (x4 and x5) to the stored variables (x1 and x2). These free variables are used during the VIR-operation so that the binding to the constituent PP and V is guaranteed. The following functional description and f-structure is produced:

S → PP V S'
    x1=x4, x2=x5, x3=x6
PP → PREP NP
    (x1 PREP) = 'mit', (x1 OBJ) = x7
NP → ART NOUN
    (x7 ART) = 'dem', (x7 HEAD) = 'Löffel'
V → VERB
    (x2 VERB) = 'aß'
S'→ NP VP
    (x6 SUBJ) = x8, x6 = x9
NP → NAME
    (x8 NAME) = 'Hugo'
VP → V NP PP
    x9 = x5, (x9 OBJ) = x10, (x9 POBJ) = x4
NP → ART NOUN
    (x10 ART) = 'das', (x10 HEAD) = 'Eis'

$$x3 = x6 = x9 = x5 = x2 = \begin{bmatrix} SUBJ = x8 \\ VERB = \text{'aß'} \\ OBJ = x10 \\ POBJ = x4 \end{bmatrix}$$

$$x8 = \begin{bmatrix} NAME = \text{'Hugo'} \end{bmatrix}$$

$$x10 = \begin{bmatrix} ART = \text{'das'} \\ HEAD = \text{'Eis'} \end{bmatrix}$$

$$x1 = x4 = \begin{bmatrix} PREP = \text{'mit'} \\ OBJ = x7 \end{bmatrix}$$

$$x7 = \begin{bmatrix} ART = \text{'dem'} \\ HEAD = \text{'Löffel'} \end{bmatrix}$$

■

5.2.5 Categorial Grammar

Categorial Grammar (CG) is an unconventional grammar theory which assigns all grammar rules to the dictionary entries, making any additional explicit grammar superfluous. There exist two kinds of categories: *basic categories* (e.g. S, N) and *complex categories* (e.g. NP/N, S\NP). The original theory [Bar-Hillel64] consists of only two combinatory rules for the formation of complex categories (A, B representing grammatical categories):

- Forward functional application: A/B B → A (5.3)
- Backward functional application: B A\B → A (5.4)

Example 5.14:

The sentence *Hugo aß mit dem Löffel (=Hugo ate with the spoon)* is parsed by use of the following grammar:

Hugo	aß	mit	dem	Löffel
NP	(S\NP)/PP	PP/NP	NP/N	N
			NP	
		PP		
	S\NP			
S				

■

The above two combinatory rules were extended by Steedman [Steedman85, Steedman87] by four rules for functional composition and type raising resulting in *Combinatory Categorial Grammar (CCG)* (see also [Dowty87, Weir88]):

- Forward functional composition: A/B B/C → A/C (5.5)
- Backward functional composition: B\C A\B → A\C (5.6)
- Type raising: B → A/(A\B) (5.7)
  B → A\(A/B) (5.8)

As an additional extension *variable categories* have been proposed [Zeevat88, Hoffman93]. This powerful generative capacity has been applied to cover some important non-canonical natural language constructions like *wh-extraction* or *nonconstituent conjunction* (e.g. see [Seiffert88, Carpenter91]). The other side of the coin is that parsing of CCG as such has been turned out to be inefficient leading to *spurious ambiguity* [Wittenburg86]. Therefore, a lot of work was done to improve the performance of CCG parsers, e.g. compiling the grammar in a *predictive form* [Wittenburg87, Wittenburg91], *normal-form based parsing* [König89, Shieber94] or *lazy chart parsing techniques* [Pareschi87]. Vijay-Shanker and Weir have proposed the *only polynomial parsing scheme* so far by using *stacking machinery* [Vijay-Shanker90, Vijay-Shanker94].

By adding features, local registers, and unification operators, CCG was extended to *Categorial Unification Grammar* (CUG) introduced by Uszkoreit [Uszkoreit86a]. Besides of syntax analysis, morphology [Hoeksema84, Whitelock88], natural language generation [Novak89], and speech processing [Steedman91] have been proposed as application fields of categorial grammars. Also some work was done on the treatment of modifiers, specifiers, and quantifiers [Bouma88, Maier88].

Example 5.15:

This grammar is in analogy to *Example 5.3*. In addition to the number, the cases are checked for agreement. The used symbols have the following semantics:

S ... syntactic features, V ... value after functional application
D ... direction of functional application, A ... argument of functional application
M ... morphological features, Z ... number, F ... case, EZ ... singular, MZ ... plural

Hugo	aß	mit	dem	Löffel
NP	(S\NP)/PP	PP/NP	NP/N	N

1. NP/N N $\rightarrow$ NP

$$\frac{\begin{bmatrix} S:\begin{bmatrix} V:[S:NP] \\ D:right \\ A:[S:N]\end{bmatrix} \\ M:\begin{bmatrix} Z:\{EZ\} \\ F:\{3\}\end{bmatrix}\end{bmatrix} \begin{bmatrix} S:N \\ M:\begin{bmatrix} Z:\{EZ,MZ\} \\ F:\{1,2,3,4\}\end{bmatrix}\end{bmatrix}}{\begin{bmatrix} S:NP \\ M:\begin{bmatrix} Z:\{EZ\} \\ F:\{3\}\end{bmatrix}\end{bmatrix}}$$

2. PP/NP NP $\rightarrow$ PP

$$\frac{\begin{bmatrix} S:\begin{bmatrix} V:[S:PP] \\ D:right \\ A:[S:NP]\end{bmatrix} \\ M:[F:\{3\}]\end{bmatrix} \begin{bmatrix} S:NP \\ M:\begin{bmatrix} Z:\{EZ\} \\ F:\{3\}\end{bmatrix}\end{bmatrix}}{\begin{bmatrix} S:PP \\ M:[F:\{3\}]\end{bmatrix}}$$

3. (S\NP)/PP PP $\rightarrow$ S\NP

$$\frac{\begin{bmatrix} S:\begin{bmatrix} V:[S:X] \\ D:right \\ A:[S:PP]\end{bmatrix} \\ M:[Z:\{EZ\}]\end{bmatrix} \begin{bmatrix} S:PP \\ M:[F:\{3\}]\end{bmatrix}}{\begin{bmatrix} S:X \\ M:[Z:\{EZ\}]\end{bmatrix}} \qquad X = \begin{bmatrix} V:[S:S] \\ D:left \\ A:[S:NP]\end{bmatrix}$$

4. NP S\NP $\rightarrow$ S

$$\frac{\begin{bmatrix} S:\begin{bmatrix} V:[S:S] \\ D:left \\ A:[S:NP]\end{bmatrix} \\ M:[Z:\{EZ\}]\end{bmatrix} \begin{bmatrix} S:NP \\ M:[Z:\{EZ\}]\end{bmatrix}}{\begin{bmatrix} S:S \\ M:[Z:\{EZ\}]\end{bmatrix}}$$

■

In order to solve the problem of constituent transfer without using type raising and functional composition, the technique of *gap-threading* has been proposed [Wesche88, Millies89] as illustrated in *Example 5.16*.

Example 5.16:

By use of gap-threading the example sentence *Mit dem Löffel aß Hugo (=With the spoon ate Hugo)* is analysed as follows:

```
Mit       dem        Löffel    aß          Hugo
PP/NP     NP/N       N         (S\NP)/PP   NP
          ---------------
                NP
------------------
       PP
                                     ⇓ GAP-insertion
                               S\NP [gap: PP]
                                     ⇓ GAP-insertion
                               S [gap: PP] [gap: NP]
                ⇓ GAP-elimination
------------------------------------------
                S [gap: NP]
                       ⇓ GAP-elimination
--------------------------------------------------
                       S
```

■

5.2.6 Summary

We gave a survey about some of the most prominent grammar formalisms: Phrase Structure Grammar, Transitions Nets, Logic Grammar, Lexical Functional Grammar, and Categorial Grammar. For each grammar its recent developments are discussed and the formalisation of sentences as well as basic parsing techniques are clarified by use of numerous examples. Special attention was given to the various techniques of dealing with constituent transfer, a linguistic phenomenon that possesses special importance for the analysis of free word order languages.

5.3 Extended Categorial Unification Grammar

As formal framework for the syntactic analysis within IDA we have selected CUG because of two main reasons [Winiwarter93c]:

- the availability of a powerful hierarchical dictionary which makes it possible to assign the grammar rules to the appropriate level of abstraction
- the bottom-up parsing strategy which is in conformity with LDL semantics and makes it possible to analyse incomplete or ill-formed sentences in a natural way

Since CUG as such is not applicable to languages with free word order like German and the solutions proposed and presented above all lack of declarative expressiveness we chose an alternative approach of extending CUG by a formalism adapted from the ID/LP rules of GPSG which have been proven adequate for this purpose [Hauenschild88, Meknavin93].

We changed the original notation for the two combinatory rules of functional application in order to be able to present the proposed extensions in a consistent way:

➷ C: A ← /B (5.9)

If category C is directly followed by B, then it can be transformed to A (forward functional application)

➷ C: A ← \B (5.10)

If category C is directly preceded by B, then it can be transformed to A (backward functional application)

We extended the Categorial Unification Grammar by the following concepts (see also [Winiwarter94]). Each extension is clarified by use of an example rule, its verbalisation and the application of the rule to an example phrase.

Example 5.17:

PREP: PP ← /NP

A prepositional phrase consists of a preposition directly followed by a noun phrase.

[PREP: auf, NP: die Maschine] → [PP: auf die Maschine] (to the machine) ■

✰ Syntactic feature restrictions can be added in brackets to the categories. (5.11)

This filter function increases significantly the selectivity of the parser during unification.

Example 5.18:

NP[-unb]: NP ← /NP[+unb]

An unknown phrase (by default assigned to basic category NP) can be joined with a known noun phrase to form a combined one.

[NP: die Maschine, NP: 7] → [NP: die Maschine 7] (the machine 7) ■

✰ New symbols '>' for indirect precedence and '<' for indirect succession. (5.12)

This covers cases of long distance dependencies where several phrases can be shifted between the two concerned categories.

Example 5.19:

TRVERB[+sep, -inf, -part]: TRVERB ← <VPREF

Separable verb prefixes can take positions far behind the verb stem if the verb form is neither infinitive nor participle.

[TRVERB: führe, NOUN: Auftrag, VPREF: durch] →
[TRVERB: führe durch, NOUN: Auftrag] (execute order) ■

☆ More than one category can be used at the right side of a rule. (5.13)

This removes the severe restriction that in a single derivation step only adjacent categories can be applied to the derivation of a new category. The several categories are applied from left to right.

Example 5.20:

NOUN: NP ← \ADJ \ART

A noun is transformed to a noun phrase if it is directly preceded by an adjective and an article.

[ART: der, ADJ: neue, NOUN: Gehalt] → [NP: der neue Gehalt]
(the new salary) ■

☆ Introduction of the asterisk symbol indicating as usual zero or more repetitions. (5.14)

This extension is introduced in order to capture recursive constructs of phrases.

Example 5.21:

NOUN: NP ← \ADJ* \ART

A noun phrase consists of an article, several optional adjectives, and a noun.

[ART: der, ADJ: neue, ADJ: monatliche, NOUN: Gehalt] →
[NP: der neue monatliche Gehalt] (the new monthly salary) ■

☆ Enclosing of optional categories in parenthesis. (5.15)

Optional categories reduce the number of required grammar rules and increase at the same time the clearness of representation.

Example 5.22:

NOUN: NP ← \ADJ* (\ART)

A noun, directly preceded by several optional adjectives and an optional article, generates a noun phrase.

[ADJ: neuer, NOUN: Gehalt] → [NP: neuer Gehalt] (new salary) ■

☆ No function symbol in front of a category to indicate free word order. (5.16)

In this situation it is only necessary that the category is present in the phrase but no further conditions on its position are made.

Example 5.23:

TRVERB[+imp]: IC ← NP[+akk] PP*

A transitive verb with mood imperative accompanied by a noun phrase with case accusative and several optional prepositional phrases creates an imperative clause, the sequence of the three components is arbitrary.

[TRVERB: storniere, PP: für den Kunden Maier, NP: den letzten Auftrag] →
[IC: storniere den letzten Auftrag für den Kunden Maier]
(cancel the last order for the client Maier) ■

5.4 Implementation

The grammar rules are inserted as arguments to the dictionary at the appropriate level of abstraction. This grammatical argument possesses the following format:

{(NewCat, Restrict, RightSide, Priority)} (5.17)

NewCat	derived category
Restrict	syntactic restrictions
RightSide	right side of grammar rule
Priority	priority value, determines the application order of the rules

The right side of the grammar rule is mapped to a list of categories:

[(Cat, Restrict, Sequence, Occurrence)] (5.18)

Cat	category
Restrict	syntactic restrictions
Sequence	sequence condition
Occurrence	occurrence condition

The possible values for Occurrence are: erforderlich (required), optional, and '*'. For Sequence the following symbols are valid: ' ', '/', '\', '>', and '<'.

Example 5.24:

The rule

TRVERB[+imp]: IC ← NP[+akk] PP*

is mapped to the following LDL argument for transitive verbs:

{(ic, {('+', imp)}, [(np, {('+', akk)}, ' ', erforderlich), (pp, { }, ' ', '*')], 1)}

In the same way, the rule

NOUN: NP ← \ADJ \ART

is mapped to this argument for nouns:

{(np, { }, [(adj, { }, '\', '*'), (art, { }, '\', optional)], 5)} ■

The parsing of an input sentence is performed in the following way. First, based on the result of lexical analysis a *basic category* is assigned to each word resulting in an appropriate list representation. Then, the grammar rules are applied to this list by use of a *bottom-up strategy*. The syntactic features of the combined structures are *unified* at each step leading to a significant reduction of derivations. Figure 11 shows an informal example of the unification process.

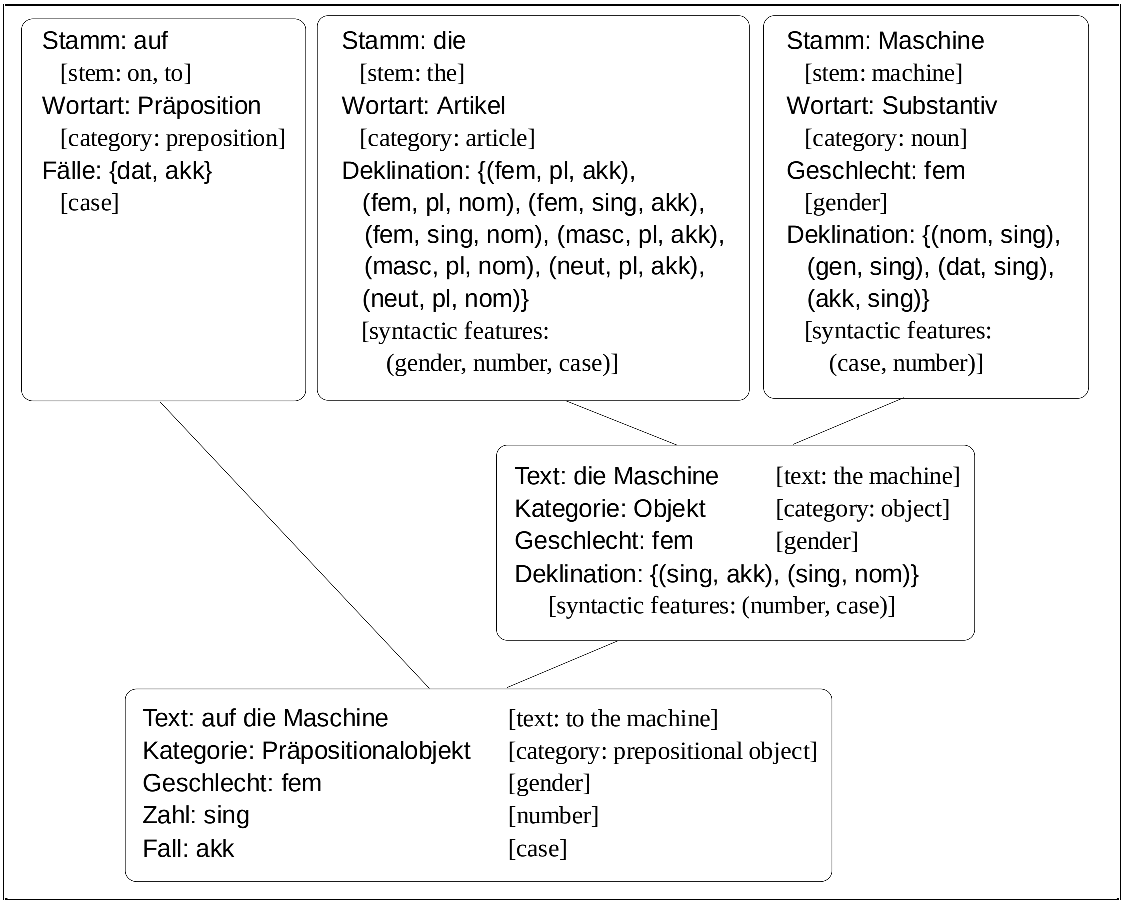

Figure 11: Example of unification of syntactic features

As already mentioned the decision which rule is applied next is not arbitrary but is determined by the priority values stored in the dictionary. The deliberate choice of these values is of crucial importance to the efficiency of the parser in that the additional effort for backtracking and trying other interpretations is minimised. Figure 12 and Figure 13 show two example segments of the LDL code. The first one represents the top level of syntactic analysis, it recursively produces all applicable rules and selects the one with the highest priority value for derivation until no more rules can be applied. The second example code checks the applicability of the right side of a single grammar rule to a specific category. Recursively, each constituent of the right side is checked against the list members. If the test holds true for the right side, the concerned list members are replaced by the new derived syntactic category.

```
analys(Liste1, Liste3) ←                         recursive rule for the syntactic analysis of an input
                                                 sentence (Liste1 ... input list, Liste3 ... output list)
    regel(Liste1, Regeln),                       generation of all applicable rules
    Regeln ~= { },                               rule set not empty
    aggregate(maxpri, Regeln, Bestregel),        determination of rule with the highest priority
    Bestregel = (Liste2, _),                     new list after application of grammar rule
    analys(Liste2, Liste3).                      next step of recursion
analys(Liste, Liste) ←                           exit rule
    regel(Liste, { }).                           triggers if no more rules can be applied

regel(Liste, Regeln) ←                           produces all applicable rules to input list Liste
    regel2(1, Liste, Liste, Regeln).             initial call of recursive generation rule

regel2(I, [X|Rest], Liste, Regeln) ←             I ... list position, X ... processed category,
                                                 Rest ... rest of list
    genregel(I, X, Liste, Regeln2),              generates set of all applicable rules for category X
    I2 = I + 1,                                  incrementing list position
    regel2(I2, Rest, Liste, Regeln3),            next step of recursion
    union(Regeln2, Regeln3, Regeln).             resulting rule sets are merged
regel2(_, [ ], _, { }).                          exit rule
```

Figure 12: LDL code segment of syntactic analysis

```
analys(Pos1, Regelliste, Liste1, Liste3) <-              recursive analysis of a single grammar rule
                                                         Pos1 ... position of considered list entry
                                                         Liste1... input list, Liste3 ... output list
                                                         Regelliste ... right side of grammar rule
      Regelliste = [Eintrag|Rest],
      analys2(Pos1, Eintrag, Liste1, Liste2),            analysis of first entry of right side
      analys(Pos1, Rest, Liste2, Liste3).                analysis of rest of right side
analys( _, [ ], Liste, Liste).                           exit rule

analys2(Pos1, (Kat, Restr, Seq, Occ), Liste1, Liste3) <- analysis of single entry of right side
                                                         Kat ... syntactic category to be searched
                                                         Restr ... syntactic restrictions
                                                         Seq ... sequence condition
                                                         Occ ... occurrence condition
      lmember(Kat, Restr, Pos2, Liste1),                 membership test yielding position in list
      if( Seq = ´<´                                      checking of sequence conditions
      then Pos1 < Pos2),
      ...
      entferne(Kat, Liste1, Liste2),                     removal of entry from input list
      if( Occ = ´*´                                      if occurrence is repetitive then
      then  analys2(Pos1, (Kat, Restr, Seq, Occ),        recursive application of rule
                              Liste2, Liste3)
      else  Liste3 = Liste2).
analys2( _, (Kat, Restr, _, optional), Liste, Liste) <-  analysis rule for optional occurrence
      ~lmember(Kat, Restr, _, Liste).                    if concerned category is absent
analys2( _, (Kat, Restr, _, ´*´), Liste, Liste) <-       exit rule for repetitive occurrence
      ~lmember(Kat, Restr, _, Liste).
```

Figure 13: Test of the applicability of the right side of a grammar rule

Finally, Figure 14 shows a simplified example (leaving out of consideration the unified features) of the syntactic analysis of an input sentence.

Example sentence:

Füge den neuen Mitarbeiter Max Huber mit Anfangsgehalt 20000 hinzu !

(Insert the new worker Max Huber with initial salary 20000 !)

Basic categories:

[TRVERB, ART, ADJ, NOUN, NP, PREP, NOUN, NP, VPREF]

[Füge, den, neuen, Mitarbeiter, Max Huber, mit, Anfangsgehalt, 20000, hinzu]

Analysis:

1) TRVERB[+sep, -inf, -part]: TRVERB ← <VPREF

[TRVERB, ART, ADJ, NOUN, NP, PREP, NOUN, NP]

[Füge hinzu, den, neuen, Mitarbeiter, Max Huber, mit, Anfangsgehalt, 20000]

2) NOUN: NP ← \ADJ* (\ART)

[TRVERB, NP, NP, PREP, NOUN, NP]

[Füge hinzu, den neuen Mitarbeiter, Max Huber, mit, Anfangsgehalt, 20000]

3) NOUN: NP ← \ADJ* (\ART)

[TRVERB, NP, NP, PREP, NP, NP]

[Füge hinzu, den neuen Mitarbeiter, Max Huber, mit, Anfangsgehalt, 20000]

4) NP[-unb]: NP ← /NP[+unb]

[TRVERB, NP, PREP, NP, NP]

[Füge hinzu, den neuen Mitarbeiter Max Huber, mit, Anfangsgehalt, 20000]

5) NP[-unb]: NP ← /NP[+unb]

[TRVERB, NP, PREP, NP]

[Füge hinzu, den neuen Mitarbeiter Max Huber, mit, Anfangsgehalt 20000]

6) PREP: PP ← /NP

[TRVERB, NP, PP]

[Füge hinzu, den neuen Mitarbeiter Max Huber, mit Anfangsgehalt 20000]

7) TRVERB[+imp]: IC ← NP[+akk] PP*

[IC]

[Füge hinzu den neuen Mitarbeiter Max Huber mit Anfangsgehalt 20000]

Figure 14: Example of syntactic analysis

Of course, the main task of syntactic analysis within a natural language interface is not to derive the syntactic correctness of an input sentence but to construct its syntactic structure in parallel. For the sentence shown in Figure 14 this structure looks as displayed in Figure 15 (not showing morphological and syntactic features in detail). The symbol c stands for a derived category whereas s signifies basic categories.

```
[(ic, c, [
        (trverb, c, [
                (trverb, s, fuege),
                (vpref, s, hinzu)
        ]),
        (np, c, [
                (np, c, [
                        (noun, s, mitarbeiter)
                        (adj, s, neuen)
                        (art, s, den)
                        ]),
                (np, s, 'Hubert Maier')
        ]),
        (pp, c, [
                (prep, s, mit),
                (np, c, [
                        (np, c, [
                                (noun, s, anfangsgehalt)
                        ]),
                        (np, s, 20000)
                ])
        ])
])]
```

Figure 15: Example of syntactic structure

5.5 Summary

In order to provide a sound theoretical framework for syntactic analysis within IDA we first presented some of the influential grammar formalisms to natural language interface design. On the basis of this survey we selected Categorial Unification Grammar as optimal basis because of two reasons. First, its requirement of assigning all grammar rules to the lexical entries which fits very well with our powerful hierarchical dictionary. Second, because its bottom-up parsing strategy is in conformity with LDL semantics and satisfies perfectly our claim to analyse also incomplete and ungrammatical sentences in an easy and natural way.

We extended CUG by six new importent concepts in order to deal with the free word order of German language in a clear and concise way. Finally, the resulting compact grammatical representation scheme was applied to the implementation of an efficient parser within our IDA architecture.

6. Semantic and Pragmatic Analysis

6.1 Introduction

Semantic analysis aims at abstracting from any linguistic phenomena at the surface level of natural language sentences in order to obtain a pure representation of the underlying meaning. The correct mapping from the *surface structure* to this semantic *deep structure* is the main obstacle to the development of successful natural language applications.

The common predecessor for most current approaches of semantic representation was *case grammar* introduced by Fillmore [Fillmore68, Fillmore77]. It defines semantic relationships on the basis of a small set of semantic roles, the so-called *cases*, which the parts of a sentence can perform. In *Section 6.2 Semantic Analysis* we give a short view of existent semantic representation techniques and strategies for the co-operation with syntactic analysis before presenting the approach chosen for our IDA architecture.

Since semantic analysis alone is in many cases insufficient for the correct interpretation of natural language sentences, world and context knowledge is incorporated in *pragmatic analysis* to eliminate semantic ambiguities. Finally, *spelling error correction* can only be performed within the framework of semantic analysis in order to be able to distinguish for example misspelled database values from new ones for insertion or update operations.

6.2 Semantic Analysis

The numerous representation schemes that have been proposed to model semantics have their origin from *knowledge representation* and can be divided in three main streams (for good surveys see [Schwind85, Luger89]):

- *Logical schemes* [Woods78, Schubert82, Hobbs87, Bollinger89] define semantics by use of logic facts and rules. Extensions to the basic deductive inference strategy are circumscription [McCarthy80], default logic [Reiter80, Yuan93], autoepistemic logic[Moore85] or abductive reasoning [Hobbs88, Brewka93, Hsu93, Rayner93].
- *Graphical schemes* were first introduced in the form of *semantic networks* [Quillian68, Woods75] where objects are represented by *nodes* and relationships by *links*. The main drawback of semantic nets was the missing standardisation as concerns the labelling of links. Therefore, *conceptual dependency* [Schank74] tried to overcome this deficiency by defining a uniform notation (for an example see Figure 16). More recent approaches are *conceptual graphs* by [Sowa84], *sentence formalism* [Binot84] or *propositional semantic networks* [Castelfranchi84, Ali93].
- *Structured schemes* are based on the notion of *frames* introduced by [Minsky75] that defines types of *objects* which store the information in so-called *slots*. Relationships are modelled by use of pointers to other frames, additional features include the specification of initial and unspecified values, conditions on creation, and declarative as well as procedural attachments. Figure 17 shows a simple example of these different types of slots. For recent extensions we refer to [Hirst82, Binot86, Wu93].

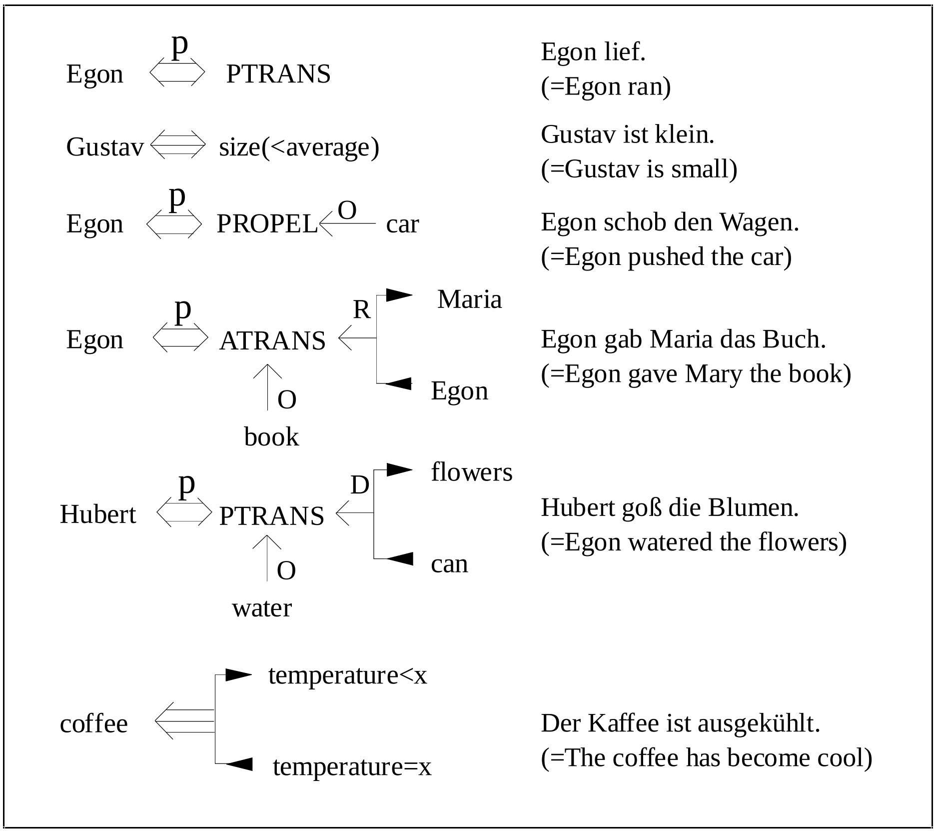

Figure 16: Example of conceptual dependency

	Student	
unspecified identifier	Register number	empty
relation to frame	Super type	*Person* (*Slots* Name, Age etc.)
unspecified attribute	Number of terms	empty
relation to frame	Studies	*Study* (*Slots* Name etc.)
initial value	Student type	full
Condition	Education	School-leaving examination
procedural attachment	Average	compute sum of all marks and divide it by their number
declarative attachment	Result	if average < 1,5 and number of mark fair <= 1, then excellent

Figure 17: Example of frame

With regard to the interaction of semantic and syntactic analysis it has turned out to be advantageous not to follow the strictly sequential process model but to overlap the two steps of analysis. This is justified by the reduction of problem space for the parser by eliminating meaningless or contradictory interpretations already at an early point of processing [Cappelli84]. Depending on the degree of interaction, the following different types of approaches exist, for each one several references of prominent implementations are given:

- *Semantic grammars* incorporate semantics directly in grammar rules, they were used by early natural language systems like SOPHIE [Brown75], PLANES [Waltz77] or LIFER [Hendrix78].
- *Rule-by-rule interpretation* performs semantic analysis of each intermediate result and makes decision about acceptance or rejection [Thompson75, Robinson82, Pereira80, Kay85, Reyle88].
- *Preference based interpretation* also analyses the result of each syntactic analysis step but gives only a rating about its semantic plausibility [Wilks75, Weischedel83, Fass83, Jensen87].
- *Interleaved parsers* restrict semantic interpretation to main constituents [Winograd72, Bobrow80, Woods80, Ritchie80, McCord85].
- *Semantically driven parsing* use syntactic analysis exclusively if necessary for disambiguation within semantic processing [Birnbaum81, Schank75, Riesbeck78, Lytinen86, Helbig86].

After carefully considering the above design choices, we judged the last approach of semantically driven analysis as most favourable solution for the development of database systems with well-defined semantic application models within the IDA architecture. This decision is strongly motivated by laying more stress upon the *information extraction* paradigm rather than upon *text understanding*. This dichotomy was introduced by Appelt [Appelt93] for the field of information filtering, he uses the following distinguishing criteria:

- *information extraction:*
  - mapping to a well-defined target representation
  - subtle nuances of meaning are of no importance, e.g. mood of user
  - input can include redundant information
- *text understanding:*
  - the meaning of extensive texts have to be grasped based on the complete context
  - the complexity of the target representation corresponds to that of natural language
  - all nuances of meaning have to be detected

Whereas the information extraction approach is well established for information filtering systems (see [Lewis92, Chinchor93], much work on natural language interfaces still contribute to the text understanding paradigm. Therefore, they suffer from serious overhead of analysis (for a critique of such systems see [Schwartz82]). Only few work exists that derive benefit from the underlying database model for semantic analysis, e.g. see [Wallace84, Hui88, Schröder88].

The main reason for this can be seen in the fact that so far for interfaces to relational databases no adequate semantic model existed or caused by loosely-coupled architectures no access was possible. Only the complete integration of the linguistic analysis within the database architecture makes it possible to merge the two representation schemes in a natural and consistent way. Therefore, the computation effort is minimised by providing at the same time a maximum of quality of analysis.

As pre-requisite of semantic analysis we assigned semantic features to the dictionary entries at the appropriate level of abstraction by making use of inheritance. For similar approaches which also use hierarchically structured dictionaries for the efficient processing of semantic features see [Flickinger85, Shieber86, Uszkoreit86b]. Figure 18 displays an example of the attachment of semantic features, also illustrating how divergent specific meanings of derived words can overwrite more general combined ones.

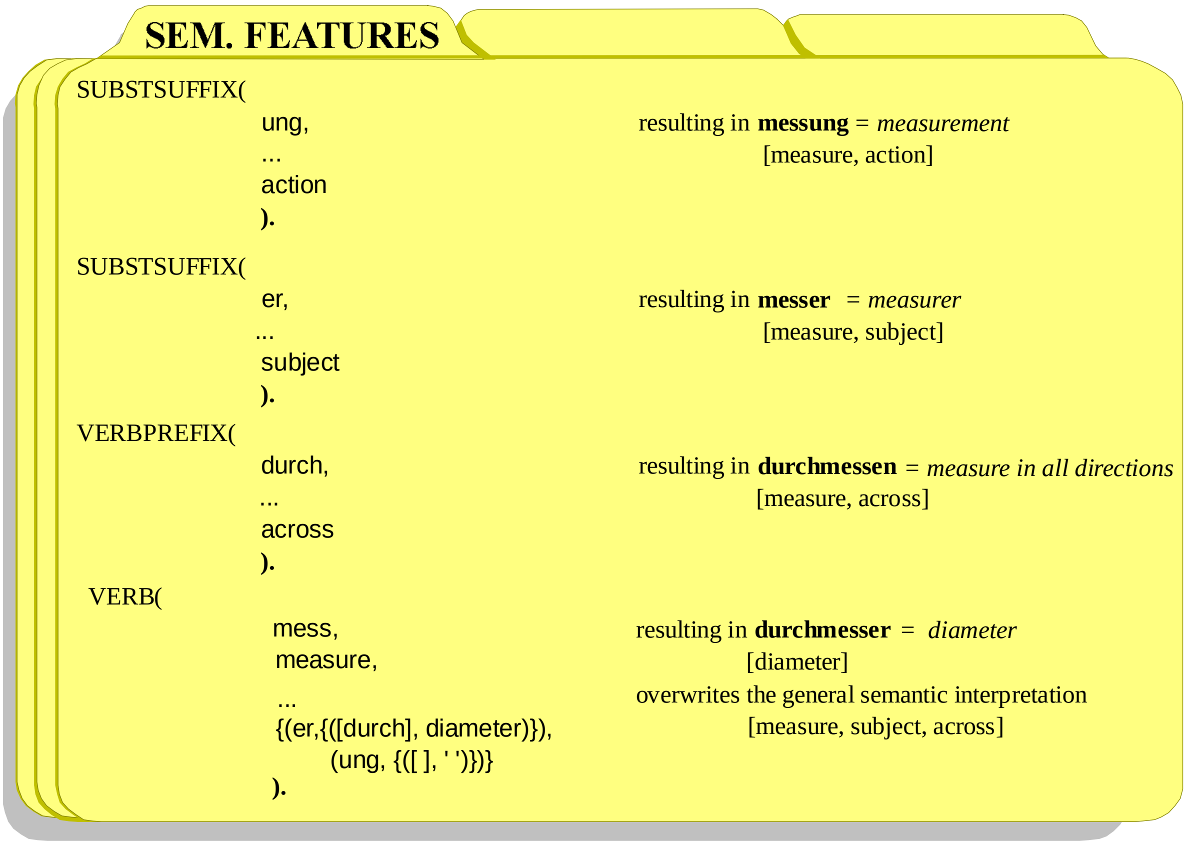

Figure 18: Example of semantic features

The morphological analysis computes for each word its *deep form* as is illustrated in Figure 19 by the LDL code for the generation of the deep form of derived substantives. It can be distinguished between four different types of general deep forms which again can be overwritten by specific deep forms:

particles: [Sem] (6.1)

adjectives and verbs: [Sem, SemPr] (6.2)

substantives: [Sem, SemPrSeq] (6.3)

derived words: [Sem | [SemSuf | SemPrSeq]] (6.4)

Therefore, the output of the morphological component takes the form of an *deep form list (DFL)* which gives for the individual input words a set of possible interpretations, each entry indicating the word stem, the word category, and the semantic deep form:

[{(Wortstamm, Kategorie, Tiefenform)}] (6.5)

Example 6.1:

For the following example sentence the corresponding DFL is computed:

Die neue Mindestbestellmenge von St 50 H ist 25 Stück
(=The new minimal order quantity of St 50 H is 25 pieces)

DFL:

[{(die, artikel, [die]), (die, relativpronomen, [die])}, {(neu, adjektiv, [neu])},
{(menge, substantiv, [menge, stell, be, mindest])}, {(von, praeposition, [von])},
{('St', unknown, string)}, {('50', unknown, integer)}, {('H', unknown, string)},
{(sein, verb, [sein])}, {('25', unknown, integer)}, {(stueck, substantiv, [stueck])}] ■

```
wort(Wort, (Eintr, ableitSubst, SemG)) <-   classifies word as derived substantive
                                             resulting in: stem, word category, deep form
   verb(Eintr,Sem,_, ...  ,_, Suffixe),      retrieval of verbs
   affixe(Wort, Eintr, Pr, Suffix),          sub-string test of verb stems (external C-predicate)
                                                 if satisfied, it returns the separated affixes
   suffixtest(Suffix, Suffixe,               checks suffix with suffixes in dictionary yielding
     Praefixe, SemSuf),                      set of valid prefixes and general semantic feature
   praefixtest(Pr, Praefixe, SemPrSeq,       checks prefix with valid prefix sequences
     SpezSem),                               yielding specific or general semantic feature
   if(SpezSem ~= ' '                         if specific semantic feature exists,
   then SemG = [SpezSem]                     then it is assigned to deep form
   else SemG =                               else deep form is constructed from the semantic
     [Sem | [SemSuf | SemPrSeq]]                   features of  verb, suffix, and prefixes
   ).
```

Figure 19: Example of LDL code for generation of deep form

An important difference of natural language interfaces in comparison to other fields of application for natural language processing techniques is the fact that unknown values possess a particular significance for the meaning of the sentence. Also in this context, only the IDA architecture makes it possible to distinguish between existent database values and new database values for insertion or update. If one considers also the misspelling of database values, the situation becomes even more complex (see *Section 6.4 Spelling Error Correction)*.

Furthermore, existent database values can serve as identifiers to *entities* and *entity types* within the database application. Again, valuable information can be obtained which reduces the number of possible interpretation and increases the efficiency of the natural language analysis.

Therefore, we propose a preliminary step for semantic analysis, the so-called *unknown value list analysis (UVL-analysis)*. Its task is to transform the DFL produced by the morphological component to the following list presentations (see Figure 20 for an example of the transformation performed on the sentence in *Example 6.1)*:

- *unknown structure list (USL):* contains all unknown values as sub-lists, that is, compound values are split up to several list entries
- *unknown value list (UVL):* compound values are joined together, strings which represent numbers are converted
- *unknown type list (UTL):* compound and string values are looked up in the dictionary, if they represent identifiers of existent entities, the corresponding entity type is indicated, otherwise the value unknown is inserted

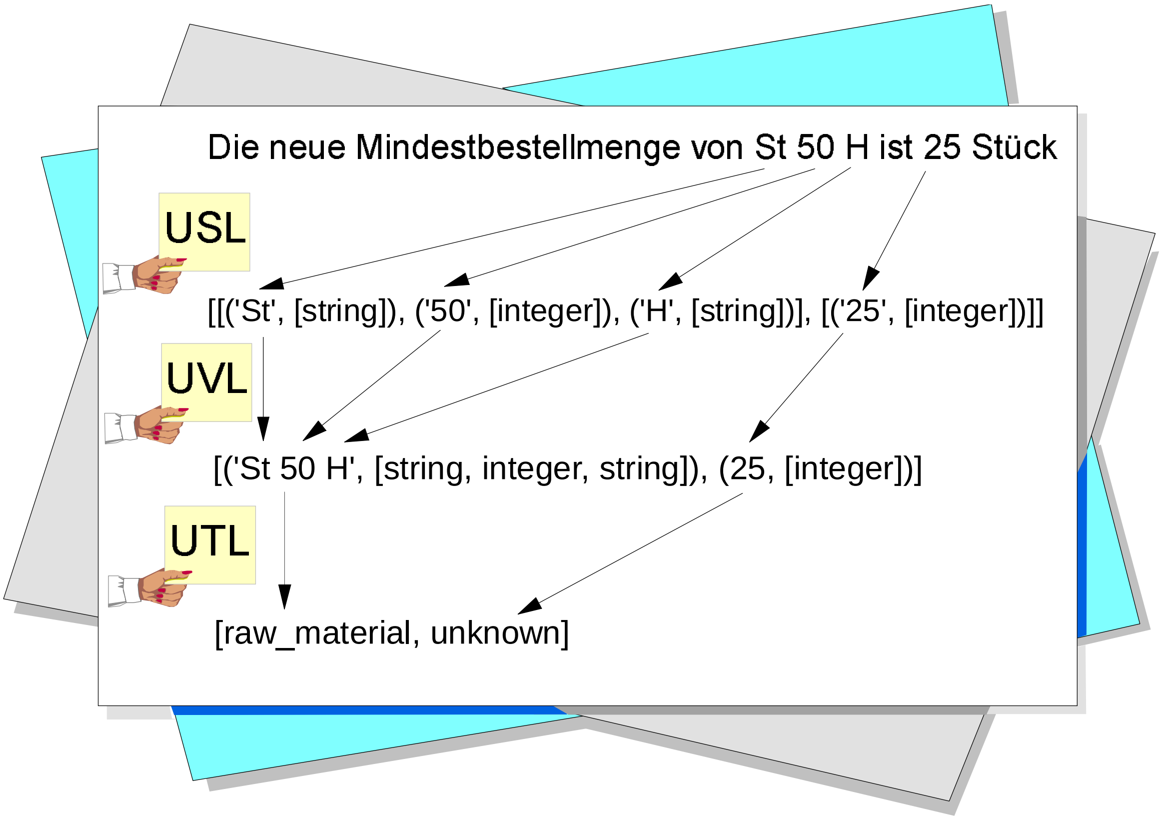

Figure 20: Example of UVL-analysis

Figure 21 displays part of the LDL code, it performs the first step of UVL-analysis, that is, the transformation of DFL to USL. The UVL-analysis forms a sound basis for efficient semantic analysis which maps the meaning of the user input to appropriate *sentence deep structures (SDS)*. These deep structures correspond exactly to the *semantic categories* of the underlying database application, therefore they guarantee the correct and efficient semantic analysis of input sentences (see *Section 7.7 Implementation of Natural Language Interface* for an implementation example and *Section 7.8 Evaluation* for evaluation data).

```
genusl(L, Ergebnis) ←                          generates USL out of DFL
    suchbeg(L, L2),                            searches for begin of unknown value
    if(L2 ~= [ ]                               if unknown value exists
    then                                       then
        zusfg(L2, Rest, Eintrag),                  create sub-list for unknown value
        genusl(Rest, Eintrag2),                    recursive call
        Ergebnis = [Eintrag | Eintrag2]            inserts unknown value into USL
    else                                       else
        Ergebnis = [ ].                            empty list is returned
genusl([ ], [ ]).                              exit rule of recursion

suchbeg([Eintrag | Rest], L) ←                 searches for next unknown value
    aggregate(auswahl, Eintrag, Eintrag2),     retrieves entry from set of interpretations
    Eintrag2=(_, Kat, _),                      retrieves category of actual entry
    if(Kat=unknown                             if category equals unknown
    then                                       then
        L=[Eintrag | Rest]                         list of remaining entries is returned
    else                                       else
        suchbeg(Rest, Rest2),                      recursive call
        L=Rest2).
suchbeg([ ], [ ]).                             exit rule of recursion

zusfg([Eintrag | Rest], Rest2,                 analysis of unknown value
        [(Wort, Typ) | Rest3]) ←
    aggregate(auswahl, Eintrag, Eintrag2),     retrieves entry from set of interpretations
    Eintrag2=(Wort, Kat, Typ),                 retrieves word stem, category, and type
    if(ntrkat(Kat)                             if no separating category
    then                                       then
        zusfueg(Rest, Rest2, Rest3)                joining parts of composed unknown value
    else                                       else
        Rest2=Rest,                                remaining categories are returned
        Rest3=[ ]).                                single unknown value is returned

zusfueg([Eintrag | Rest], Rest2, Ergebnis) ←   parts of unknown value are composed
    aggregate(auswahl, Eintrag, Eintrag2),     retrieves entry from set of interpretations
    Eintrag2=(Wort, Kat, Typ),                 retrieves word stem, category, and type
    if(fs(Kat, Typ, Rest)                      if criteria for continuation are satisfied
    then                                       then
        zusfueg(Rest, Rest2, Rest3),               recursive call
        Ergebnis=[(Wort, Typ) | Rest3]             result is computed
    else                                       else
        Rest2=[Eintrag | Rest],                    unknown value is added to remaining entries
        Ergebnis=[ ]).                             empty list is returned to start composition
zusfueg([ ], [ ], [ ]).                        exit rule of recursion
```

Figure 21: LDL code for generation of USL

6.3 Pragmatic Analysis

Generally speaking, pragmatic analysis aims at improving the quality of semantic analysis by making use of knowledge beyond the scope of the analysed sentence. This can involve on the one hand some kind of *meta-knowledge* or *world knowledge*, on the other hand knowledge about the embedding *context* (for a good general survey see [Allen87]).

For world knowledge the delimitation to semantic analysis is rather fluid, also the applied techniques again originate from the area of knowledge representation (see *Section 6.2 Semantic Analysis)*. A common definition is that pragmatics goes beyond the *domain knowledge* of the application model and includes some kind of 'common sense', e.g. subtle nuances of meaning expressed by stylistic choice [Makuta-Giluk93].

Context knowledge takes in natural language interfaces the particular form of *dialogue* or *discourse* between the user and the computer. This is also the main reason why semantic analysis alone will in many cases fail to produce a correct interpretation of a user command since the user assumes that the system 'remembers' the topic of the preceding queries. Therefore, efficient natural language interfaces should include the capability of *discourse resolution*, that is, to supplement the missing information by keeping track of the prevailing conversation in order to prevent the user of the boring task of repeating again and again the same pieces of information (for good surveys see [Hirst81, Frederking88a]). The most influential theoretical framework represents the *Discourse Resolution Theory (DRT)* which was introduced by Kamp in 1981 [Kamp81, Kamp88], for recent extensions see [Frederking88b, Bras90]. There also exists the promising approach to extend CUG by DRT constructs [Zeevat88].

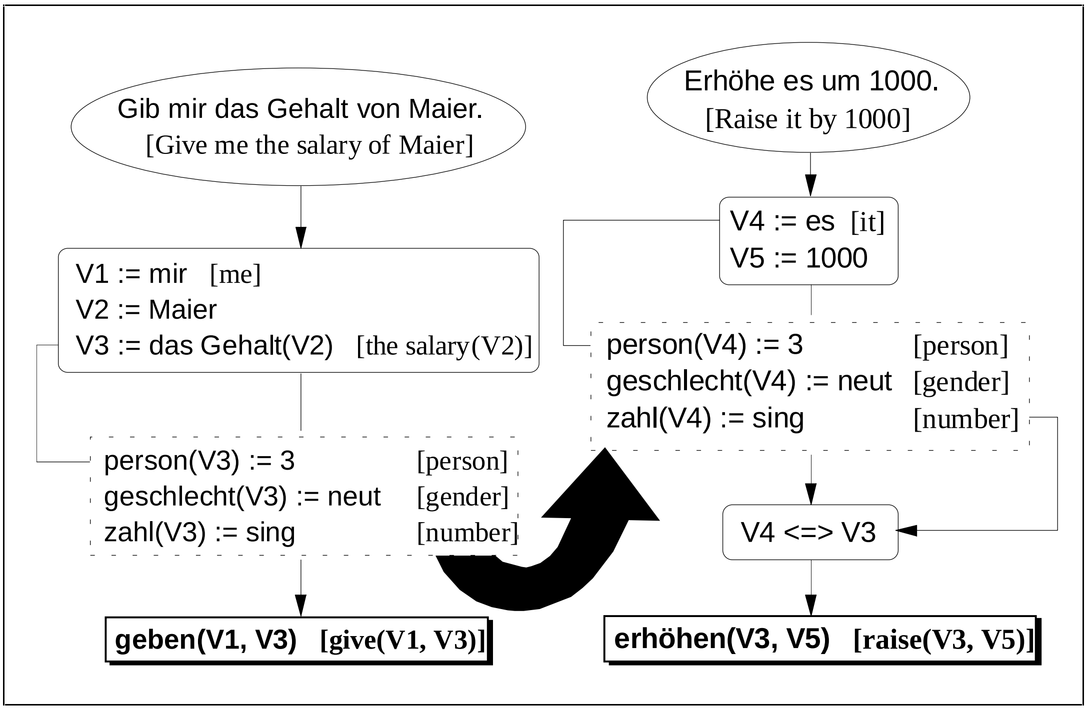

Figure 22: Example of de-referencing anaphora

The two main linguistic phenomena which *discourse resolution* has to deal with are *ellipses* and *anaphora*. Whereas ellipses are incomplete sentences without any obvious hint about the omitted data, anaphora represent references to so-called *antecedents* which can be de-referenced more easily [Webber83], see Figure 22 for an example using DRT notation. Besides simple *history list* techniques, there exist also more sophisticated approaches which apply *non-monotonic logic* [Dunin-Keplicz84], *coherence relations* [Kehler93] or *abductive reasoning* [Nagao93a].

A lot of mainly theoretical work was done about the use of world knowledge for discourse resolution. The applied methods have their origin in the analysis of *stories*, they model *actions*, that is, characteristic sequences of situations. Prominent representation techniques are *scripts* [Schank77] and *plans* [Wilensky83, Allen80]. Figure 23 illustrates the modelling of an action by means of an easy example script for an ice-cream parlour.

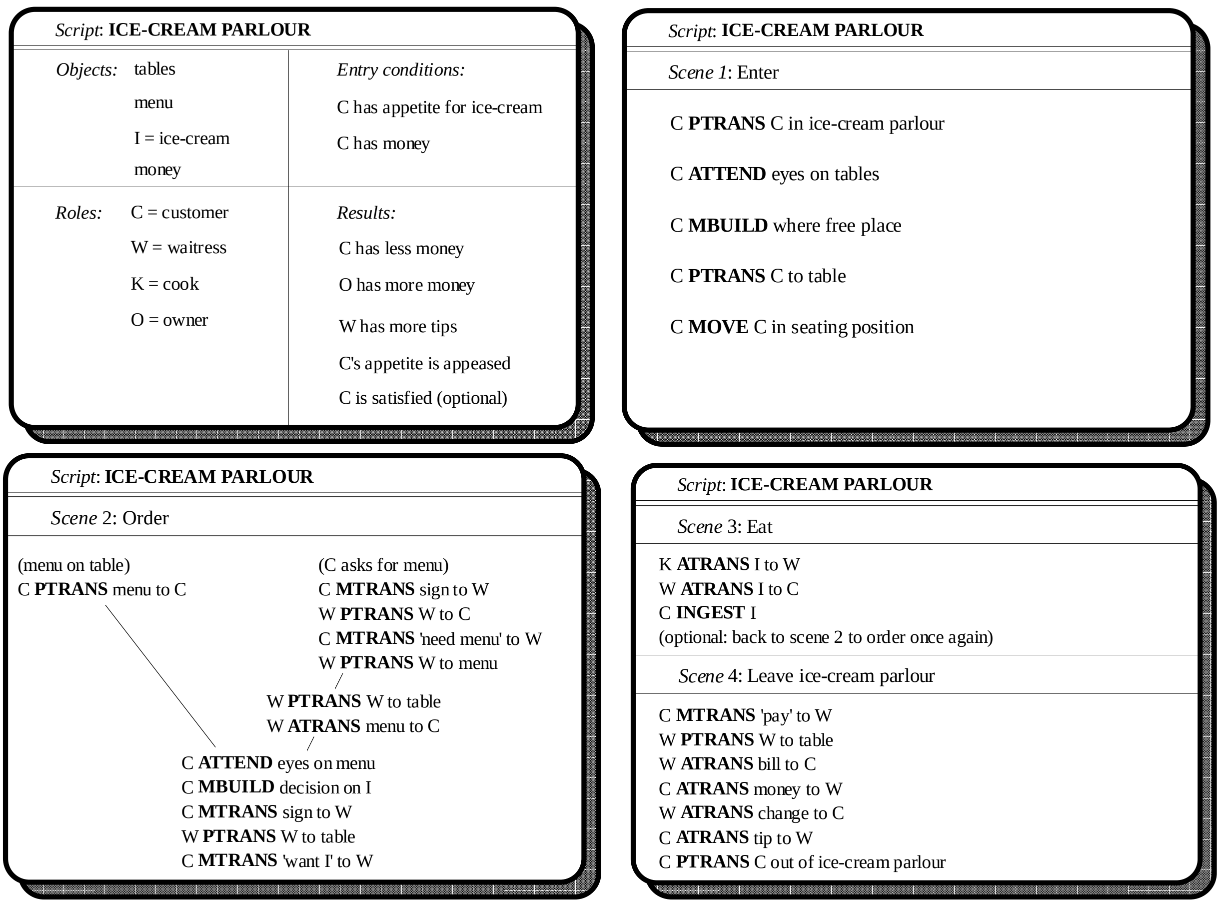

Figure 23: Example of script

Plans, which possess a richer flexibility in comparison to scripts, have proven to be adequate for modelling *speech acts* (see [Cohen79, Allen80, Perrault80], recent extensions were contributed by [Cohen85, Litman87, Grosz86, Carberry89, Haller93]). Finally, also the distinction of the user's belief from general knowledge has been the subject of intensive research which has its origin in automated reasoning (for basic theoretical work on belief modelling see [Hintikka69, Kripke63, Moore73, Cohen78], more recent developments can be found in [Levesque84, Steel84, Fritsch85, Ramsay87, Ghose93].

For the pragmatic analysis in our IDA architecture we concentrated our research on the resolution of ellipses and anaphora. For that purpose we used a *uniform semantic resolution method (USRM)* which abstracts from the syntactic surface structure. For each successful analysed input sentence, the entity as well as the corresponding entity type is extracted from the semantic deep structure and inserted to the deductive database as LDL base predicate. This actual focus of the user session is then applied to the resolution of ellipses and de-referencing of anaphora.

This technique turned out to be very efficient and, in spite of its simplicity, capable of analysing all of the occurring discourse resolution problems correctly (see *Section 7.8 Evaluation)*. The main reason for this is on the one hand the sequential nature of database sessions. On the other hand, the pre-requisite in order to achieve this satisfactory performance is the quality of semantic analysis with regard to the homogenous mapping to the semantic model of the underlying application.

Example 6.2:

Figure 24 displays an example where in the first situation data about the product *Schraubstock* (=vice) is requested whereas in the second case 7 pieces are ordered by customer *Anton Huber* without indicating the name of the product. Therefore, the last entity *Schraubstock* with the correct entity type *Produkt* is substituted properly. ■

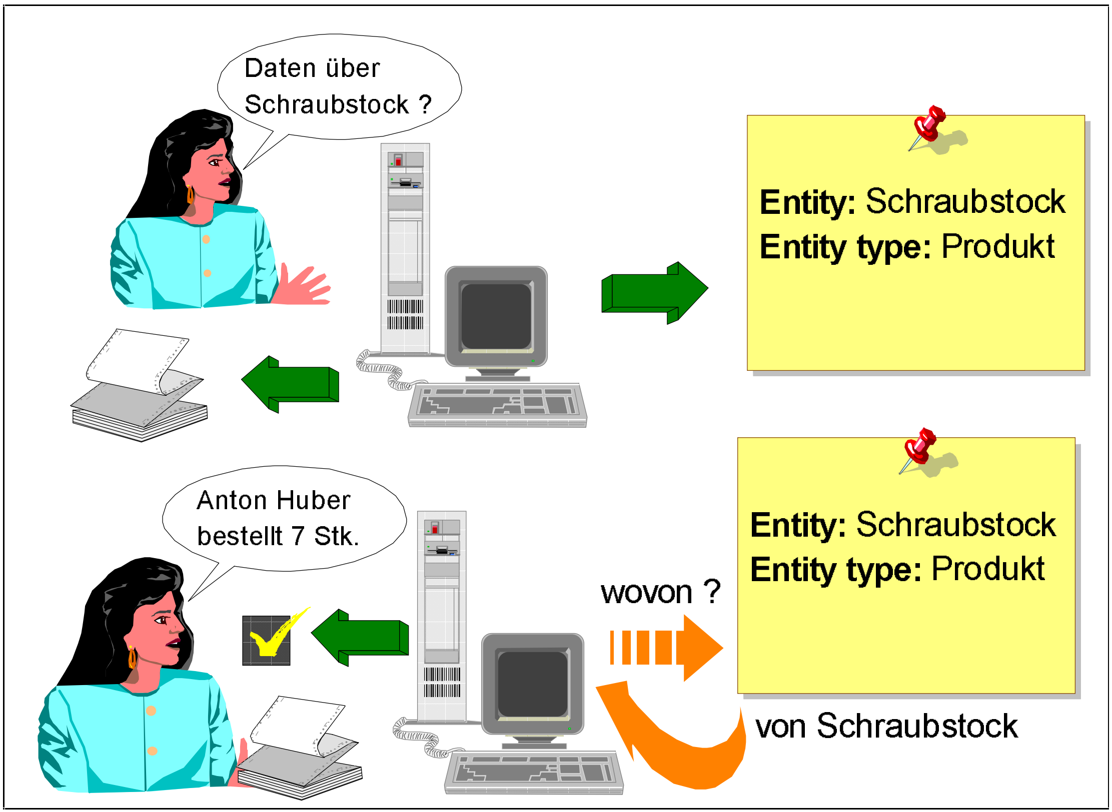

Figure 24: Example of uniform semantic resolution method

6.4 Spelling Error Correction

The existence of misspelled words is one of the main obstacles to the correct analysis of real-word input sentences. This is especially true for the field of natural language interfaces because of the additional difficulty of distinguishing misspelled words from new database values for insertion or update (see [McFetridge90]). Therefore, spelling error correction can only be applied in connection with semantic analysis (see *Section 6.2 Semantic Analysis*).

With regard to the type of the misspelled word three different situations can occur:

- ☹ misspelled function words
- ☹ misspelled existent database values
- ☹ misspelled new database values

Whereas for the third case of course no possibility of error detection exists, the other two types can be eliminated in principle. However, as functional terms are only stored as canonical forms, the correction algorithm involves complex computations, thus significantly reducing the efficiency of analysis. So far only simple methods for suffix-tree approaches dealing with very restricted error types (e.g. single omission, insertion, substitution or interchange at adjacent positions [Dorffner85]) have been proposed. Another approach is the matching of the word in question with so-called *n-grams* which represent valid character sequences of length *n* in order to calculate the type and position of the error, e.g. [Mundt88].

Since the second category, the misspelling of existent database values, is much more likely to appear in database interfaces and can cause severe consequences, we concentrated our research on this error type. The higher frequency rate is also motivated by the fact that misspelled database values do not have as only reason user mistakes but are also caused by the intentional use of inflections.

The basic idea to correct faulty user input is the comparison with existent database values in order to decide if similar entries exist. Therefore, the central issue of spelling error correction is to find an appropriate measure which represents the similarity between two words. The only similarity measure so far that considers the special characteristics of database values was introduced by Bickel [Bickel87]. It possesses the following specific properties:

- ➢ computation is based on the number of equal letters
- ➢ the order of the letters is arbitrary
- ➢ each character is only counted once

However, the proposed method possesses the following severe drawbacks:

- 👎 the erroneous doubling of characters is not detected
- 👎 insensitivity as concerns additional wrong characters, this is a substantial weak point especially for technical applications, e.g. similar parts with long initial common sub-strings
- 👎 only alphabetic characters are considered which is again not practicable for technical data
- 👎 no standardised interval for the similarity value

Therefore, we introduced four important adaptations and improvements:

- each common occurrence of a character is rated positively
- each divergent occurrence of a character is rated negatively
- numbers and special characters are included
- the similarity value is standardised on the interval [-1; +1], i.e. it equals +1 in the case of identity and -1 if the opposite is true

In order to compute the similarity value, we developed the following formula which satisfies all of the specified properties:

$$SIM = \frac{\sum_{i=1}^{k}\left[3 \cdot \min\left(z_{i1}, z_{i2}\right) - \max\left(z_{i1}, z_{i2}\right)\right]}{\sum_{i=1}^{k} z_{i1} + \sum_{i=1}^{k} z_{i2}} \qquad (6.6)$$

In this formula $z_{i1}(z_{i2})$ signifies the number of occurrences for the character i in the first (second) word. The numerator shows the similarity or difference between the two terms whereas the standardisation of the resulting value is taken care of by the denominator. In *Example 6.3* and *Example 6.4* we illustrate the semantics of this formula by means of a pair of very similar and a pair of very divergent words.

Example 6.3:

The German word *Plandrehen* (=to face) is compared with the erroneous word *Plandehen* (omission of one character).

Plandrehen:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Σ
1	0	0	1	2	0	0	1	0	0	0	1	0	2	0	1	0	1	0	0	0	0	0	0	0	0	10

Plandehen:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Σ
1	0	0	1	2	0	0	1	0	0	0	1	0	2	0	1	0	0	0	0	0	0	0	0	0	0	9

Differences:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Σ
2	0	0	2	4	0	0	2	0	0	0	2	0	4	0	2	0	-1	0	0	0	0	0	0	0	0	17

Similarity:

$$SIM = \frac{17}{10+9} = \frac{17}{19} = 0{,}89$$ ■

Example 6.4:

The German word *Augenlagerteil* (=lug-bearing part) is compared with the different word *Schraubstock* (=vice).

Augenlagerteil:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Σ
2	0	0	0	3	0	2	0	1	0	0	2	0	1	0	0	0	1	0	1	1	0	0	0	0	0	14

Schraubstock:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Σ
1	1	2	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	2	1	1	0	0	0	0	0	12

Differences:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Σ
1	-1	-2	0	-3	0	-2	-1	-1	0	-1	-2	0	-1	-1	0	0	2	-2	2	2	0	0	0	0	0	-10

Similarity:

$$SIM = \frac{-10}{14+12} = \frac{-10}{26} = -0,38$$ ■

In the following we will prove that the similarity value is really mapped to the interval [+1; -1]. For the case of identical words $z_{i1}=z_{i2}=z_i$ holds true for all characters *i*. Therefore, *(6.6)* becomes to:

$$SIM = \frac{\sum_{i=1}^{k}\left[3 \cdot \min(z_i, z_i) - \max(z_i, z_i)\right]}{\sum_{i=1}^{k} z_i + \sum_{i=1}^{k} z_i} = \frac{2 \cdot \sum_{i=1}^{k} z_i}{2 \cdot \sum_{i=1}^{k} z_i} = 1 \qquad (6.7)$$

In the opposite extreme case that the two terms have no characters in common at all, for each character *i* either z_{i1} or z_{i2} equals *0*. For the first case, the argument of the sum in the dominator of *(6.6)* is reduced to:

$$z_{i1} = 0\colon\ 3 \cdot \min(z_{i2}, 0) - \max(z_{i2}, 0) = -z_{i2} \qquad (6.8)$$

By analogy it equals $-z_{i1}$ for the second case so that the sum in the dominator can be split up in two partial sums leading to the required result:

$$SIM = \frac{\sum_{i=1}^{k}(-z_{i1}) + \sum_{i=1}^{k}(-z_{i2})}{\sum_{i=1}^{k} z_{i1} + \sum_{i=1}^{k} z_{i2}} = -1 \qquad (6.9)$$

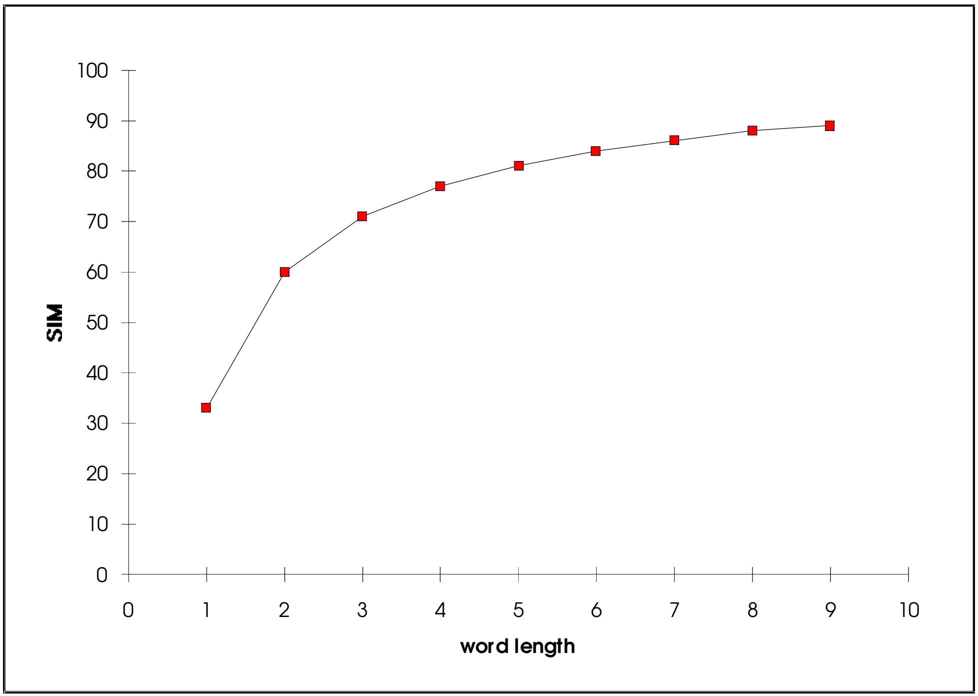

Figure 25: Similarity value for insertion of one character

Figure 25 and Figure 26 show the behaviour of the similarity value with regard to different error types. The insertion of one divergent character is shown in Figure 25 as function of the word length. Of course, for the removal of a character the characteristics is the same. As the interchange of characters does not effect the similarity value at all, only the substitution of a character has to be considered as additional error type. Figure 26 shows that the gradient angle is smaller compared with the case of wrong insertions.

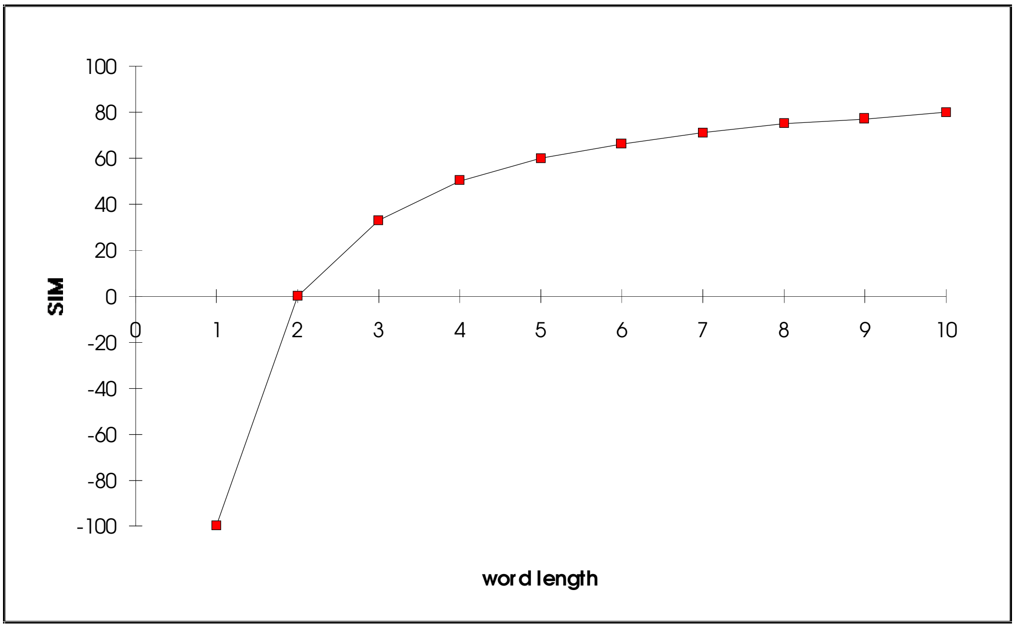

Figure 26: Similarity value for substitution of one character

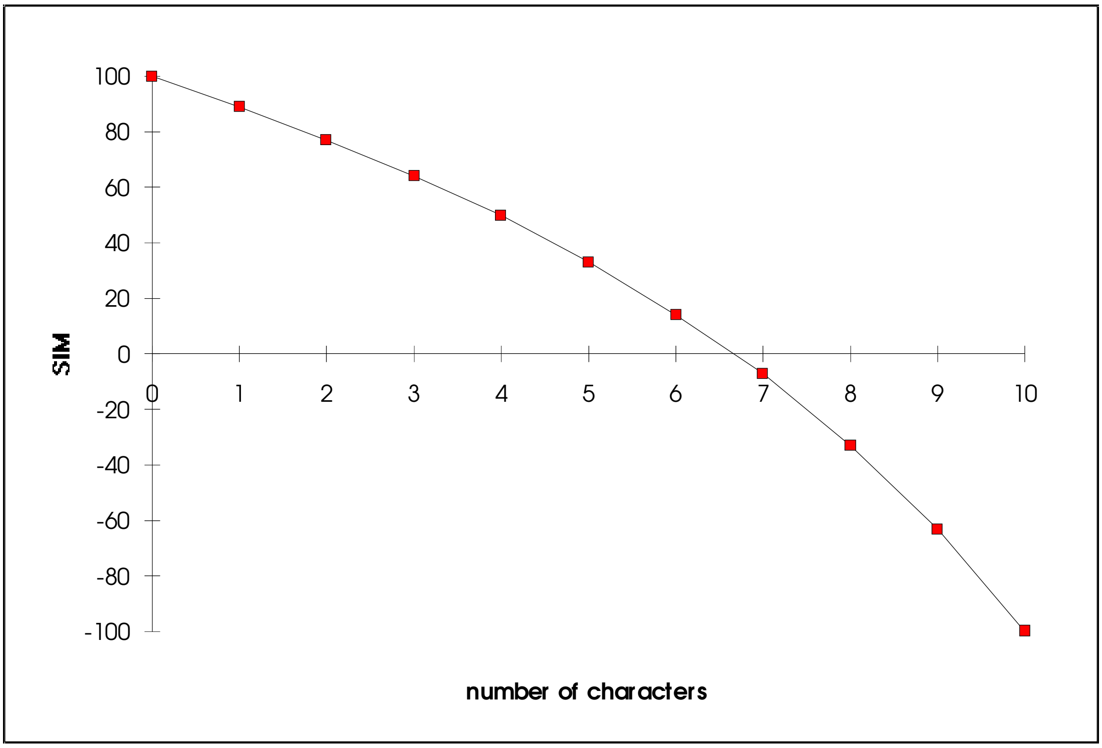

Figure 27: Similarity value for deletion of several characters

As second type of behavioural analysis we used a fixed word length of 10 for the first word and varied the number of errors as concerns the second word. Figure 27 and Figure 28 compare the effects of several deletions or substitutions. Whereas in the first case the resulting graph is convex, for the latter situation a linear relation is displayed.

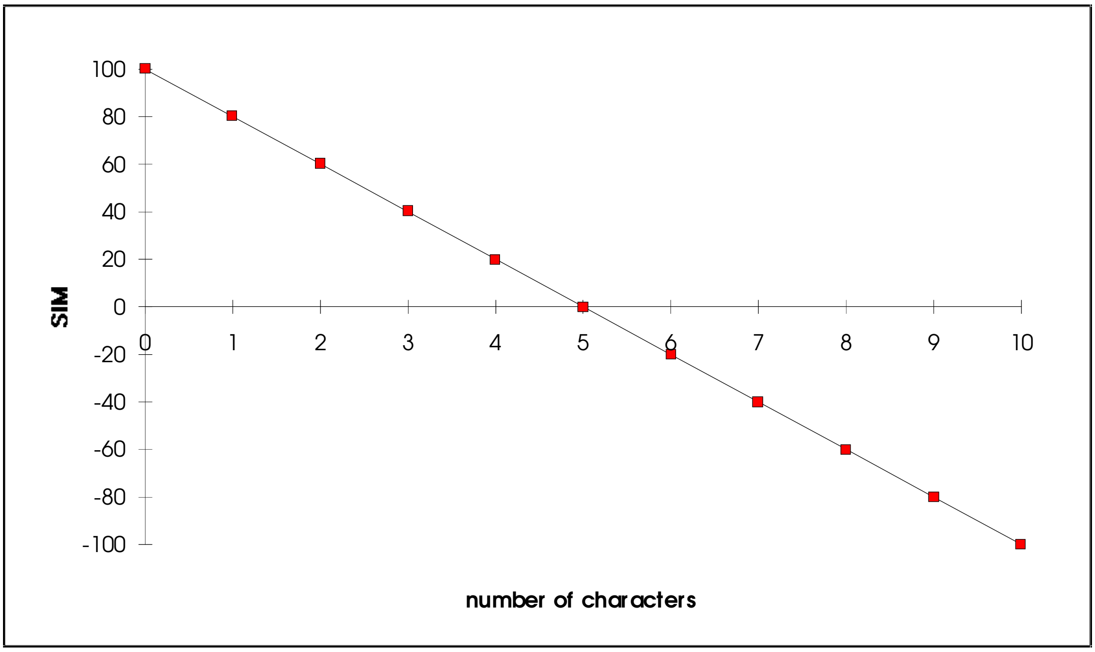

Figure 28: Similarity value for substitution of several characters

Due to the complete integration of the natural language interface and the application database, the LDL rule for realising spelling error correction is relatively simple and straightforward. If semantic analysis finds no solution for a sentence containing unknown words by interpreting them as new database values (see *Section 6.2 Semantic Analysis)*, the unknown strings are substituted by the most similar database values before semantic analysis is applied again.

```
suchobj2([(W1, Wt) | R1], [Ot1 | Rt1],            recursive rule which corrects UVL and UTL
        [(W2, Wt) | R2], [Ot2 | Rt2]) <-
  if(Ot1 = unknown                                if entity type is unknown,
        v(Wt),                                       entity identifier is string,
        suchbegr(W1, B, K)                           and there exists similar entity,
  then W2 = B,                                    then entity
        Ot2 = K                                      and entity type is substituted
  else W2 = W1,                                   else they remain unchanged
        Ot2 = Ot1),
  suchobj2(R1, Rt1, R2, Rt2).                     recursive call of rule
suchobj2([ ], [ ], [ ], [ ]).                     exit rule of recursion

suchbegr(Suchwert, Begriff, Kat) <-               retrieval of similar entity and entity type
  suchbegD(Suchwert, M),                          retrieval of set of most similar entities
  cardinality(M, 1),                              test for unique maximum
  member((Begriff, Kat), M).                      retrieves entity and entity type from set

suchbegD(Suchwert, <(Begriff, Kat)>) <-           most similar entities are grouped
  suchbegC(Suchwert, Begriff, Kat).

suchbegC(Suchwert, Begriff, Kat) <-               computes most similar entities
  suchbegA(Suchwert, M),                          set of similar entities
  suchbegB(Suchwert, M2),                         set of corresponding similarity values
  aggregate(max, M2, Maxwert),                    computes maximum of similarities
  member((Begriff, Kat, Maxwert), M).             retrieves entities with maximum similarity

suchbegB(Suchwert, <(Begriff, Kat, Wert)>) <-     grouping of similar entities
  suchbeg(Suchwert, Begriff, Kat, Wert).

suchbegA(Suchwert, <Wert>) <-                     grouping of similarity values
  suchbeg(Suchwert, _, _, Wert).

suchbeg(Suchwert, Begriff, Kat, Sim) <-           retrieves similar entities
  schwellwert(Limit),                             threshold value
  objkat(Begriff, Kat),                           retrieves entities
  compare(Suchwert, Begriff, Sim),                C-predicate for computing similarity value
  Sim > Limit.                                    similar if similarity value > threshold
```

Figure 29: LDL rule for spelling error correction

In order to retrieve a candidate for substitution, first the set of similar terms is computed by use of the similarity value (calculated by an external C-predicate) and a pre-defined threshold. Out of this set, the entity with the maximum similarity is selected for substitution.

Figure 29 displays the rule for the recursive search for similar entities which corrects the entries in UVL and UTL, that is, it substitutes the entity in UVL and enters the corresponding entity type in UTL. As can be seen, throughout the rules also the unlikely situation that there is no unique maximum (because there are two very similar entities with exactly the same similarity) is considered, in this case no substitution is performed.

The threshold value represents a crucial parameter because it determines the cardinality of the set of similar terms. If on the one hand it is selected too high, the number of corrected failures is reduced. On the other hand, a value chosen too small can result in erroneous replacements. For these reasons the threshold value is realised by a base predicate in the deductive database which can be updated freely in order to adjust the spelling error correction to the specific needs of the actual application.

6.5 Summary

We discussed in this Section the critical touchstone for each natural language interface, the ability of correctly interpreting the intended meaning of user input. Semantic analysis deals with this problem by mapping the surface structure of an input sentence to an appropriate deep structure. We provided a short survey of representation techniques and explained how the dictionary is supplemented by semantic features in IDA. With regard to the interaction with syntactic analysis different architectural solutions were considered carefully before introducing the UVL-approach for database applications with well-defined semantic models, a technique that is based on the evaluation of database values and deep forms of functional words. Therefore, no complete grammatical sentence structures are computed but syntactic information is only used if necessary for disambiguation.

Since manipulation or retrieval of data is seldom performed by use of a single command but rather takes the form of a dialogue between user and computer, a great deal of research was done in pragmatic analysis aiming at extending the scope of analysis to the complete user session. For that purpose, many sophisticated theories and models have been proposed in literature, most of them also incorporating world knowledge beyond the scope of the application domain. We concentrated our research on pragmatic analysis in IDA to discourse resolution by introducing a simple but efficient uniform semantic resolution method (USRM) which abstracts from specific manifestations at the surface level (ellipsis, anaphora) by using the entity and entity type of the preceding analysis to keep track of the actual focus.

Finally, we dealt with spelling error correction, one of the most important features as concerns user acceptance by preventing the user of the tedious task of retyping erroneous input. In this context, IDA performs an optimal basis for the correction of misspelled database values because of the complete integration of the application data within the natural language interface. This makes it possible to verify efficiently the erroneous input word with the existent entries in order to retrieve a candidate for substitution. For the crucial point of estimating the likeliness between two terms we introduced a new formula for the calculation of similarity values and analysed its properties thoroughly.

7. Case Study: PPC Database for Precision Tools

7.1 Introduction

As field of application for our case study we have chosen a *production planning and control system (PPC)* as nucleus for a later extension to a full CIM system ([Scheer84, Scheer87], for a more recent survey also dealing with security aspects see [Vieweg94]). The main components of a CIM system and their mapping to appropriate database support is shown in Figure 30. As the complexity of CIM applications requires advanced database functionality [Kappel94], they represent a challenging area for the use of deductive database technology.

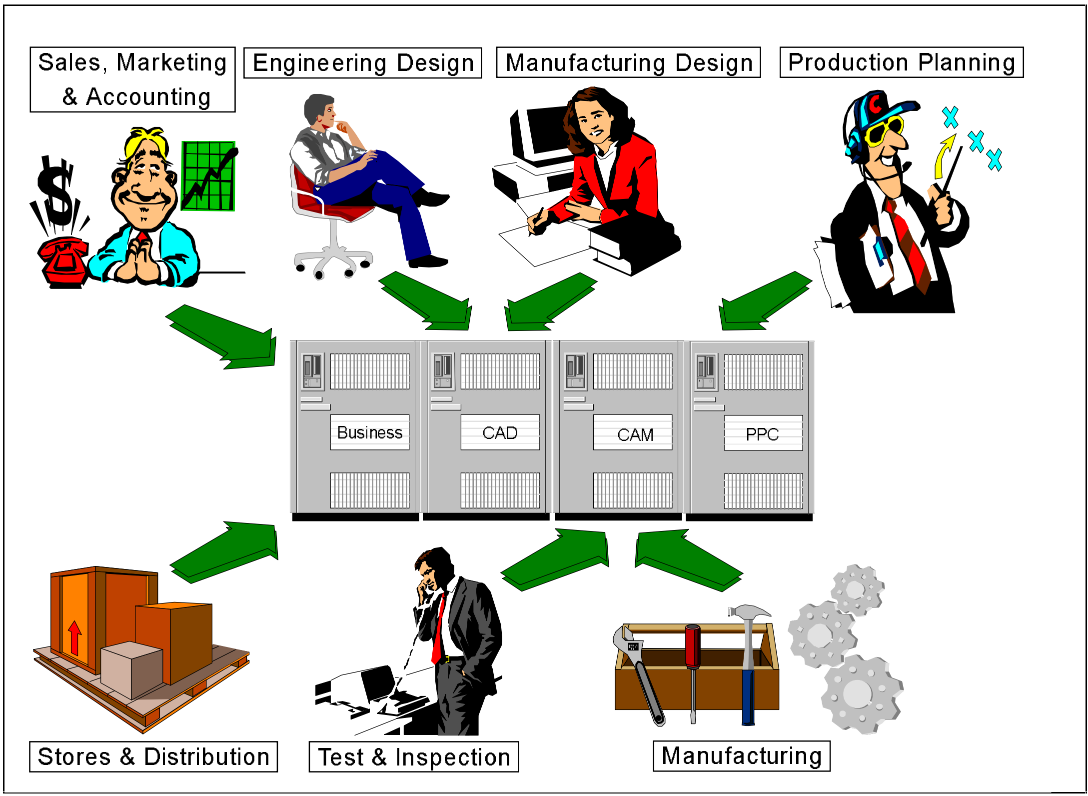

Figure 30: Database support in CIM

The PPC performs the mean-term scheduling of products and involved resources in the manufacturing processes shown in Figure 31, that is, material, machines, and labour (see [Baetge84]). The resulting master production schedule forms the basis for the co-ordination of related business services such as engineering, manufacturing, and finance.

For the complex task of planning the optimal sequence of machining operations and of assigning operators and machines to the individual machining operations, techniques from operations research are needed [Isermann79, Küpper84]. Finally, due to unforeseen circumstances (machine stoppages, operator drop outs, delivery delays) the control component of the PPC bears vital importance for the immediate reaction to such sudden changes in the availability of resources.

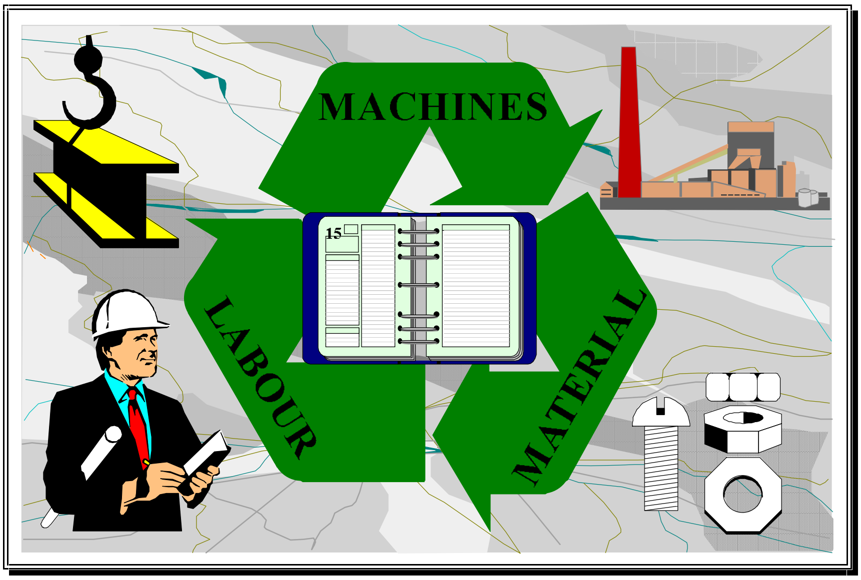

Figure 31: Types of resources

The modelled enterprise makes precision tools using as basic strategies job order production and serial manufacture. Especially in this branch of industry there exists the strong need of modelling complex objects (e.g. the assembly of a part) and transitive relations such as operation sequences or sub-part hierarchies. As the efficient realisation of these demands exceeds the power of relational database technology, the application presents an excellent choice for deriving full advantage of the extended functionality of deductive database systems. Furthermore, the sophisticated functionality justifies the effective use of a natural language interface.

For the design and implementation of the PPC system, we apply the following *seven step model.* As initial step we define in *Section 7.2 Requirements Analysis* the requirements to the PPC system in full detail. By the use of the Extended Entity Relationship (EER) model [Navathe83, Elmasri85] we then transform in *Section 7.3 Database Definition* the verbal description to a corresponding conceptual model which serves as sound basis for the transformation to appropriate LDL base predicates.

The next step in *Section 7.3 Specification and Implementation of the Functionality* covers the functionality of the PPC system. In order to obtain a well-defined reference model for the development of the natural language front-end, we specify exactly 50 manipulations and 50 queries to the PPC which are implemented by LDL rules. Finally, the semantics of the functional part is formally represented as deep structures in *Section 7.5 Semantic Application Model*.

As starting point for the implementation of the natural language interface, questionnaires are used in *Section 7.6 Empirical Collection of Test Data* to get 1000 realistic example sentences (10 for each command). On the basis of this data, the interface is implemented in *Section 7.7 Implementation of Natural Language Interface* and finally the whole application is evaluated in *Section 7.8 Evaluation*, i.e. the correct mapping of the 1000 surface structures to the 100 deep structures as well as examples of spelling error correction and de-referencing of anaphora are tested extensively.

7.2 Requirements Analysis

The first step of the design process was to collect and express thoughtfully the requirements to the PPC. Because of the complexity of the application, we divided the problem in two separate views. The *static view* considers only the *master data*, i.e. static entities and stable relationships that are not likely to change during a medium-term perspective whereas the *dynamic view* models the *transaction data*, that is, transitory entities and short-term relations. Throughout the verbal description we used italics to identify entity and relationship names.

7.2.1 Static View

The objects which are machined during production are called *parts* and are divided in *assembled parts* and *basic parts.* An assembled part is composed of several parts (sub-parts), these sub-parts can again be assembled from other sub-parts and so on. Therefore, each assembled part possesses a *sub-part hierarchy* which determines its structure.

The basic parts are obtained from *suppliers*. There exist four categories of basic parts:

- *Standard parts* are ordered as prefabricated parts in standardised sizes and qualities. They are charged by piece, attributes are the name and the standard.
- *Cast parts* are manufactured by use of special moulds so that they are again charged by piece. Attributes are the name and the original material.
- *Round parts* are cylindrical basic parts which are not ordered by the name but by the *raw material* and the specification of the crude size, the charging is performed by volume.
- *Angular parts* are square prismatic basic parts, they are ordered the same way as round parts.

Therefore, as concerns supplies one can distinguish between orders for standard parts, cast parts, and raw materials which are all subsumed by the generic term *material.* For the correct processing of orders of materials the following information is of importance: the purchase price, the minimal quantity, and the time of delivery. Finally, the attributes of the suppliers are name and address.

In order to be able to perform a machining operation on a part, a *machine* and an *operator* is required. A machine is identified by its machine number (automatically created) and *machine type*, an operator by its name, social security number, address, salary, and the *qualification* which machine types he can handle. During production scheduling the machines and operators are assigned to machines, these assignments are stored in scheduling lists.

If a machine or an operator has to be removed from the production, there must be the possibility to consider this situation early in production scheduling. This guarantees that no more machining operations are assigned to the machine or operator. The status of the machine or worker is then no longer ready for operation but suspended.

The machining operation is the smallest unit in the production, its main characteristic is the kind of *action* which is determined by the required machine type as well as cost and time data. With regard to the latter it is distinguished between the fixed and the variable portion. Finally, an *operation sequence* can be assigned to each part which defines the *order of execution* of the individual actions for this part.

A *product* is the result of a production process in response to a product order by a *customer*, i.e. the products constitute a subset of the assembled parts. Important characteristics of the products are the name, minimal output, profit margin, cancellation fee, and the net selling-price. To calculate the gross selling-price, the actual *VAT rate* is needed. Finally, for the customers again the name and address is stored as information.

A vital component for each PPC is the conception of time. The temporal dimension of all future events is modelled as interval (difference of time when event occurs minus presence). The delivery time of basic parts and completion time of products are calculated by days whereas the start and completion time of machining operations are given by hours and days, a day counting as 8 hours. Therefore, the actual *time of the day* in hours has to be stored for the exact calculation of the arrival time of a delivery or the completion time of a product, the time of the day for these two events is supposed to be 0 (daybreak).

7.2.2 Dynamic View

Orders for basic parts (the order number is generated automatically by the system) are made in relation to specific product orders within the framework of production scheduling. Due to the existence of minimal quantities for deliveries, excess basic parts can occur for which a *stock of basic parts* is established. The stock can be divided in two different types: projected stocks (not yet delivered) and actual stocks (delivery already arrived). If a basic part is needed for a production order, the stock is checked first. Only if the stock is not sufficient, the missing quantity is ordered.

By analogy, for excess products a *stock of products* is kept. The stock of products also consists of projected stocks (production not yet finished) and actual stocks (production already finished). If a customer makes a *product order* (automatically generated product order number), it is first attempted to satisfy the request *from stock of products*. Only if the stock is insufficient, a production order is generated for the total quantity. As there exists also the possibility that a customer cancels a product order, the status of a product order can be: from stock, production or cancelled. The restriction of a minimal output can again result in excess products which are put in stock of products.

As first step of the production scheduling the inventories of the required basic and assembled parts are determined yielding the production list of the order which gives the information about the sequence of all necessary *machining operations*. After preparing the basic parts, either *from stock of basic parts* or by delivery, for each machining operation of the production list the earliest start time is derived from the availability of the required parts (part itself or sub-parts).

In the next step, all scheduling lists of machines and operators are looked up for valid intervals, that is, later than the earliest start time and longer than the duration of the machining operation. From all possible combinations of machine and operator assignments, the one is selected which guarantees the earliest completion time of the machining operation. Therefore, the availability time after the final machining operation represents the total production time of the product.

With regard to the control of disturbances machine stoppages, operator drop outs, and delivery delays are considered. In such a situation, first all directly or indirectly involved machining operations are detected and inserted in a change list. Then, all entries in the change list are scheduled anew. If a machine stoppage or operator drop out occurs during the execution of a machining operation, also this operation is added to the change list with a new duration (remaining processing time plus fixed time of action).

The net selling-prices are calculated on the basis of the total cost of production times the profit margin. The price calculation is not performed for each individual production order. That means that products are sold by list prices which are updated periodically to adjust them to the actual product prices.

In analogy to the production scheduling the cost calculation first determines the inventories of the necessary basic and assembled parts as well as the production list. The inventory is the basis of the material costs whereas the tooling costs can be computed from the production list (fixed costs are related to the minimal output). Finally, the sum of material costs and tooling costs yields the total cost of production.

7.3 Database Definition

7.3.1 Static View

The EER-diagram in Figure 32 shows the conceptual model of the static view of the PPC. The mapping to the according logical model (i.e. the LDL base predicates) is performed in analogy to the transformation rules of the *logical relational design methodology* (LRDB) [Teorey86].

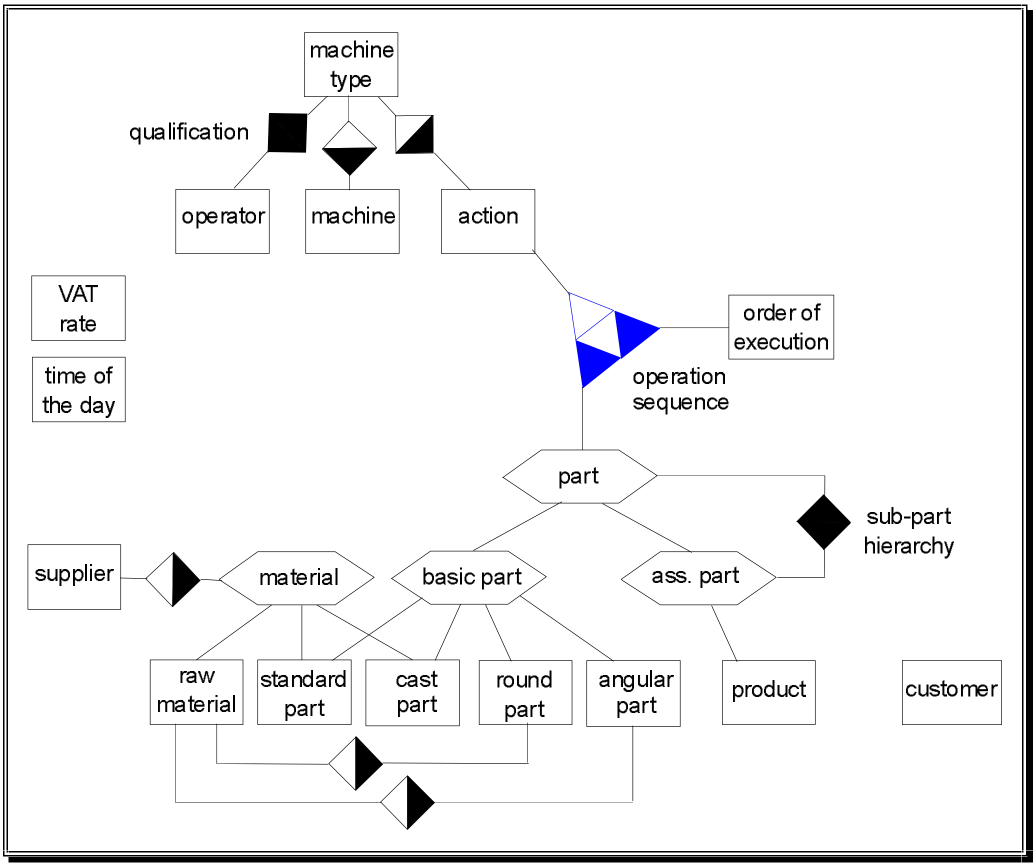

Figure 32: EER model of static PPC view

Figure 33 shows the final result of the design process. Please take notice of the following remarks while interpreting the base predicates:

- ☞ the aggregations *part*, *basic part*, and *material* are not materialised but are represented by use of derived predicates
- ☞ the relationship *sub-part hierarchy* is assigned to the entity *assembled part* as list of sub-parts, for each list member the name and the required quantity is indicated
- ☞ for the entities *supplier*, *operator*, *customer* an internal, automatically generated number is used as identifying property
- ☞ the complex structure of the scheduling list is explained in *Section 7.3.2 Dynamic View*
- ☞ the *qualification* of an *operator* is modelled as set of *machine types*, the latter are again not materialised but are represented as derived predicates
- ☞ the ternary relation *operation sequence* is transformed to an entity with the machined *part* and the list of *actions* as attributes

```
🗁 assembled_part(Name: string, Subparts: [(integer, string)]
🗁 standard_part(Name: string, Standard: string, Purchase_price: real,
     Supplier: integer, Minimal_quantity: integer, Time_of_delivery: integer)
🗁 cast_part(Name: string, Original_material: string, Purchase_price: real,
     Supplier: integer, Minimal_quantity: integer, Time_of_delivery: integer)
🗁 round_part(Name: string, Raw_material: string, Diameter: integer,
     Length: integer)
🗁 angular_part(Name: string, Raw_material: string, Length: integer,
     Width: integer, Height: integer)
🗁 raw_material(Name: string, Purchase_price: real,
     Supplier: integer, Minimal_quantity: integer, Time_of_delivery: integer)
🗁 supplier(Number: integer, Name: string, Address: string)
🗁 machine(Machine_number: integer, Name: string, Status: integer,
     Scheduling_list: [(integer, integer, integer, string, string, (integer, integer),
     (integer, integer))])
🗁 operator(Number: integer, Name: string, SSN: string, Address: string,
     Salary: real, Qualification: {string}, Status: integer, Scheduling_list:
     [(integer, integer, integer, string, string, (integer, integer),
      (integer, integer))])
🗁 action(Name: integer, Machine_type: string, Fixed_time: real,
     Variable_time: real, Fixed_cost: real, Variable_cost: real)
🗁 operation_sequence(Part: string, Actions: [string])
🗁 product(Name: string, Minimal_output: integer, Profit_margin: real,
     Cancellation_fee: real, Net_selling_price: real)
🗁 customer(Number: integer, Name: string, Address: string)
🗁 vat_rate(VAT_rate: real)
🗁 time_of_the_day(Time_of_the_day: integer)
```

Figure 33: LDL base predicates of static PPC view

7.3.2 Dynamic View

The conceptual model of the dynamic view of the PPC in Figure 34 shows only these entities of the static part that are relevant to short-time relationships, all others as well as the aggregation aspect are left out of consideration.

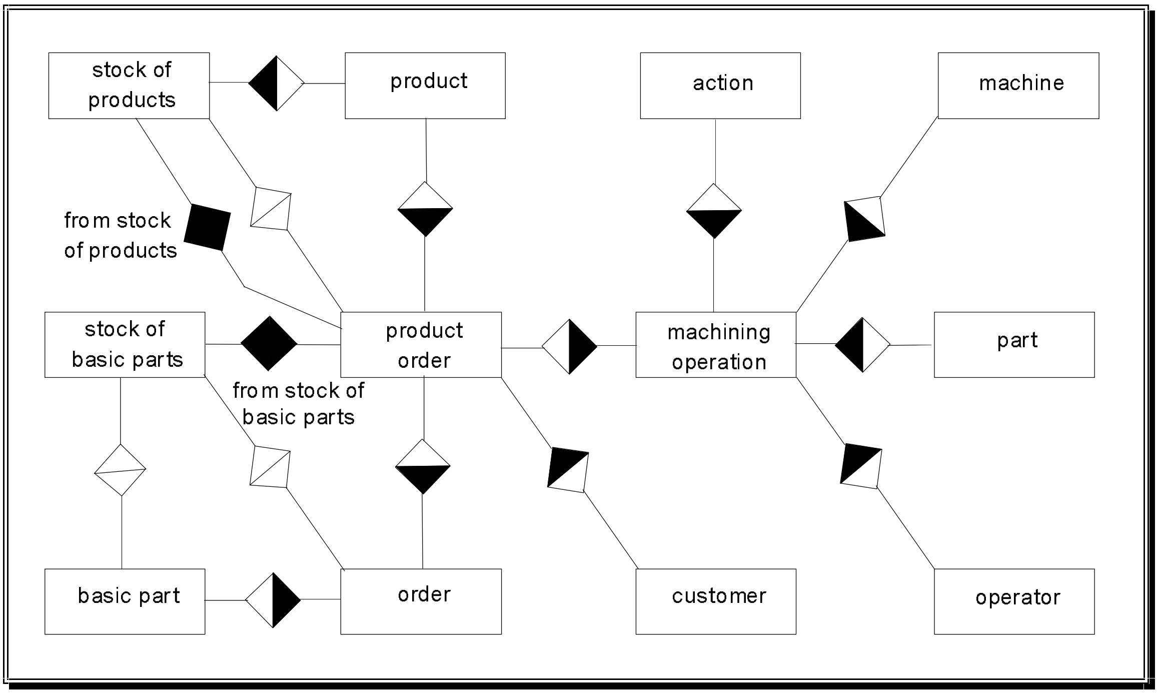

Figure 34: EER model of dynamic PPC view

The following clarification to the EER model will be helpful:

- since each *product order* first empties the *stock of basic parts* before it makes a new *order*, only one stock exists for each *basic part* (1:1 relationship between *stock of basic parts* and *basic part*)
- for the same reason the stock of a basic part has always been ordered by only one specific order (1:1 relationship between *stock of basic parts* and *order*)
- the relationship *from stock of basic parts* indicates which product orders have removed items from a projected stock, this is especially important for the correct treatment of delivery delays
- for one *product* more than one *stock of products* can exist because it is possible that several production orders with different production times produce excess products simultaneously (as shown by the 1:1 relationship between *product order* and *stock of products*)
- therefore, the connectivity of the removal of products is n:m (relationship *from stock of products)*

Again, this EER model forms the basis for the mapping to corresponding LDL base predicates shown in Figure 35 and the following annotations:

- the status of the *stock of basic parts* is actual if the delivery time is 0, otherwise it is projected
- as to the restriction of minimal output, the output of *product order* can be different from the ordered quantity, the difference represents the excess products
- the relationship *from stock of products* is mapped to corresponding lists in the entities *stock of products* and *product order* (for each entry the product order, the quantity, and the production time is given), additionally an attribute for the removal from the actual stock is appended to *product*
- removed items *from stock of basic parts* are identified by the order number by which they have been ordered
- the *machining operation* is the basic unit of production scheduling, it is not represented as separate entity but stored in the production list of the *product order* and in the scheduling lists of *machines* and *operators* by providing the following information for each list entry:

production list:	scheduling list:
⇨ sequence number of production	⇨ product order
⇨ level of sub-part hierarchy	⇨ level of sub-part hierarchy
⇨ number in operation sequence of part	⇨ number in operation sequence of part
⇨ machined part	⇨ machined part
⇨ required quantity of machined part	⇨ performed action
⇨ performed action	⇨ day and hour of start time
⇨ list of sub-parts	⇨ day and hour of completion time

```
stock_of_basic_parts(Name: string, Order: integer, Delivery_time: integer,
    Quantity: integer, Purchase_price: real)

order(Order_number: integer, Product_order: integer, Basic_part: string,
    Delivery_time: integer, Quantity: integer, Purchase_price: real)

product_order(Product_order_number: integer, Status: integer,
    Product: string, Production_time: integer, Quantity: integer,
    From_actual_stock: integer, From_proj_stocks: [(integer, integer, integer)],
    Output: integer, Net_selling_price: real, Customer: integer,
    Production_list: [(integer, integer, integer, string, integer, string, [string])])

stock_of_products(Name: string, Actual_quantity: integer,
    Projected_stocks: [(integer, integer, integer)])

from_stock_of_basic_parts(Order: integer, Product_order: integer)
```

Figure 35: LDL base predicates of dynamic PPC view

7.4 Specification and Implementation of the Functionality

Based on the specified requirements and the developed database definition we defined 50 manipulations (M1-M50) and 50 queries (Q1-Q50) to the PPC which cover the complete functionality of the modelled PPC system. The 100 commands were implemented by use of LDL rules as query forms to the deductive database and mapped to an appropriate semantic application model.

7.4.1 Manipulations to the PPC

The insertion of the following entities for the static view of the PPC is provided if the stated conditions are satisfied, otherwise the user gets a corresponding error message as response. The identifying entity numbers are generated automatically in increasing order.

Insertion of:

- ✰ *supplier* (M1)
- ✰ *raw material* (M2)
  - ➢ supplier must exist
- ✰ *standard part* (M3)
  - ➢ supplier must exist
- ✰ *cast part* (M4)
  - ➢ supplier must exist
- ✰ *round part* (M5)
  - ➢ raw material must exist
- ✰ *angular part* (M6)
  - ➢ raw material must exist
- ✰ *assembled part* (M7)
  - ➢ sub-parts must exist
- ✰ *product* (M8)
  - ➢ corresponding assembled part must exist
  - ➢ net selling-price is calculated as total cost of production times profit margin
- ✰ *customer* (M9)
- ✰ *machine* (M10)
- ✰ *operator* (M11)
- ✰ *action* (M12)
  - ➢ required machine type must exist
  - ➢ operator must exist who can handle the machine type
- ✰ *operation sequence* (M13)
  - ➢ part must exist
  - ➢ actions must exist

Similarly, under the restriction that the given conditions hold true, the following entities can be removed from the PPC. Please regard that operation sequences are deleted together with the corresponding parts. To delete a product only means that the assembled part in question is not sold anymore, the assembled part itself has to be deleted separately.

Deletion of:

- ✰ *supplier* (M14)
  - ➢ supplier must not deliver any basic part
- ✰ *basic part* (M15)
  - ➢ it must not exist a stock
  - ➢ the basic part must not be included in assembled parts
  - ➢ if a operation sequence exists, it is also deleted
- ✰ *raw material* (M16)
  - ➢ raw material must not be contained in basic parts
- ✰ *assembled part* (M17)
  - ➢ assembled part must not be included in other parts
  - ➢ assembled part must not be a product
  - ➢ if a operation sequence exists, it is also deleted
- ✰ *product* (M18)
  - ➢ it must not exist a stock
  - ➢ it must not exist any current product orders
- ✰ *machine* (M19)
  - ➢ it must not exist any assignments to machining operations
  - ➢ there must be other machines of the same machine type if the machine type is required for any action
- ✰ *operator* (M20)
  - ➢ it must not exist any assignments to machining operations
  - ➢ there must be other operators who can handle the same machine types if the machine types are required for any action
- ✰ *customer* (M21)
  - ➢ it must not exist any current product orders
- ✰ *action* (M22)
  - ➢ action must not be included in any operation sequence

With regard to the modification of master data, the following update operations on attributes of static entities are considered:

- ✰ *materials:*
  - ➢ purchase price (M23)
  - ➢ minimal quantity (M24)
  - ➢ delivery time (M25)
  - ➢ supplier (must exist) (M26)
- ✰ *suppliers:*
  - ➢ address (M27)
- ✰ *products:*
  - ➢ minimal output (M28)
  - ➢ profit margin (M29)
  - ➢ cancellation fee (M30)
- ✰ *value added tax rate* (M31)
- ✰ *operators:*
  - ➢ address (M32)
  - ➢ salary (M33)
  - ➢ insertion of machine type which the operator can handle (M34)

- *customers:*
  - address (M35)
- *actions:*
  - fixed time and cost portion (M36)
  - variable time and cost portion (M37)

Finally, some information from the static view and all entities of the dynamic view of the PPC must not be manipulated directly but can only be affected exclusively by the following important transactions:

- shift of the presence by a given interval to the future, for each event that occurs in this period a corresponding message has to be produced (M38)
- liquidation of the actual stocks of basic parts and products (M39, M40)
- calculation of selling-prices based on actual total cost of production (M41)
- scheduling of new product orders (M42)
- processing of machine stoppages and operator drop outs (M43, M44)
- processing of delivery delays (M45)
- release of machines and operators that are sooner available than expected (M46, M47)
- suspension of machines and operators (M48, M49)
- cancellation of product orders, output is transferred to the stock of products (M50)

7.4.2 Queries to the PPC

The first set of indicated queries retrieves master or transaction data. For each query it is possible to ask for the attributes of a specific entity as well as for a list of all existent ones.

- sub-parts for assembled part (Q1)
- master data of standard part (Q2)
- master data of cast part (Q3)
- master data of round part (Q4)
- master data of angular part (Q5)
- master data of raw material (Q6)
- master data of supplier (Q7)
- stock of basic part (Q8)
- order data (Q9)
- product order data (Q10)
- machine data (Q11)
- assignment data of operator (Q12)
- master data of operator (Q13)
- master data of action (Q14)
- operation sequence of part (Q15)
- master data of product (Q16)
- stock of product (Q17)
- master data of customer (Q18)
- master data of basic part (Q19)

As second query type the user can state criteria for selecting a subset of entities. These selection attributes make it possible to query the modelled relationships (see Figure 32 and Figure 34 in *Section 7.3 Database Definition).* Again, a list of all entities can be retrieved using the criteria to group the entries:

- master data of standard parts by supplier (Q20)
- master data of cast parts by supplier (Q21)
- master data of round parts by raw material (Q22)
- master data of angular parts by raw material (Q23)
- master data of raw materials by supplier (Q24)
- order data by product order (Q25)
- order data by basic part (Q26)
- product order data by product (Q27)
- product order data by customer (Q28)
- machine data by machine type (Q29)
- assignment data of operators by machine type (Q30)
- master data of actions by machine type (Q31)
- master data of basic parts by supplier (Q32)

More complex symmetric transitive relations which can be indirectly derived from the relationships of the conceptual model are covered by the following set of queries. There also exists again the choice between selection and grouping of the query result.

- assignment of production orders to orders and vice versa (Q33)
- assignment of products to customers and vice versa (Q34)
- assignment of actions to operators and vice versa (Q35)
- assignment of assembled parts or products to suppliers and vice versa (Q36)
- assignment of machine types to assembled parts or products and vice versa (Q37)

Finally, the last set of queries provides some special grouping operations, hierarchical structure information, and other special data of which the derivation requires sophisticated computation:

- stock of basic parts or products grouped by stock type (actual or projected) (Q38, Q39)
- product orders grouped by status (from stock, production or cancelled) (Q40)
- machine data and assignment data of operators grouped by status (ready for operation or suspended) (Q41, Q42)
- production time of product (Q43)
- all events which will occur in a given time interval (Q44)
- sub-part hierarchy of assembled part (Q45)
- inventory of assembled part (Q46)
- production list of part (Q47)
- difference between list price and actual product price (Q48)
- cost distribution of part (Q49)
- assembled parts or products in which a part is directly or indirectly included (Q50)

7.4.3 Implementation

In order to obtain a well-defined interface to the natural language front-end we specified 100 prototypes of query forms. The input to the requested demand as well as the resulting output are modelled by use of parameters, i.e. covered and free variables. In general all manipulations have the uniform parameter Error which returns an error code for dealing with invalid input, it equals 0 if the execution was successful. All queries are answered not as several solutions to the query form but all entries are grouped to form a unique set that is returned in the variable Result as uniform result of the query.

In the following we will give for each group of manipulations and queries a representative example of the corresponding prototype and its implementation by the use of LDL rules.

insertion of an action (M12):

```
m12($Name, $Machine_type, $Fixed_time, $Fixed_cost, $Var_time, $Var_cost, Error)
```

```
m12(Name, Machine_type, Fixed_time, Fixed_cost,
    Var_time, Var_cost, Error) ←
  if(action(Name, _, _, _, _, _)                if action already exists, then Error=1
  then Error=1
  else  if(~machine(_, Machine_type, 0, _)      if no machine with Status=0 (ready for operation)
        then  Error=2                           for action exists, then Error=2
        else  if(~can_handle(Machine_type)      if no operator can handle action, then Error=3
              then Error=3
              else Error=0,
                   +action(Name, Machine_type, Fixed_time,   otherwise action is inserted
                     Var_time, Fixed_cost, Var_cost)))).
can_handle(Machine_type) ←                      checks if any operator can handle machine type
  operator(_, _, _, _, _, Qualification, 0, _), retrieves all operators which are ready for operation
  member(Machine_type, Qualification).          test if machine type is included in any qualification
```

deletion of an assembled part (M17):

```
m17($Name, Error)
```

```
m17(Name, Error) ←
  if(~assembled_part(Name, _)                   if assembled part does not exist, then Error=1
  then Error=1
  else  if(product(Name, _, _, _, _)            if assembled part is product, then Error=2
        then Error=2
        else  if(assembled_part(_, List),       retrieves all assembled parts
                 lmember((_, Name), List)       if assembled part is included in other part,
              then Error=3                          then Error=3
              else  Error=0,
                    -assembled_part(Name, _),   otherwise assembled part is deleted
                    -operation_sequence(Name, _)))).   and corresponding operation sequence is deleted
```

update of address for customers (M35):

```
m35($Name, $Address, Error)
```

```
m35(Name, Address, Error) ←
  if(~customer(Number, Name, _)                 if customer does not exist, then Error=1
  then Error=1
  else Error=0,
       -customer(Number, _, _)                  else customer is deleted
       +customer(Number, Name, Address)).       and again inserted with new value for address
```

cancellation of product orders (M50):

```
m50($Prod_order, Error)

m50(Prod_order, Error) ←
  if(~product_order(Prod_order, _, _, _, _,_, _, _, _, _, _)     if product order does not exist, then Error=1
  then Error=1
  else  if(product_order(Prod_order, 3, _, _, _,_, _, _, _, _, _) if product order already cancelled, then Error=2
        then Error=2
        else  Error=0,
              cancel(Prod_order))).                               else product order is cancelled

cancel(P) ←                                                       cancellation of product order
  product_order(P, Status, Part, Time, Quant, FromAct,            retrieves product order
    FromProj, Output, Price, Customer, ProdList),
  if(Status=1                                                     if product order is from stock then
  then if(stock_of_products(Part, ActQuant, ProjStock)              if stock of products exists, then
        then ActQuant2 = ActQuant + FromAct,                          actual stock is returned
              retstock(FromProj, ProjStock, ProjSt2),                 projected stocks are returned
              -stock_of_products(Part, _, _),                         old stock is deleted
              +stock_of_products(Part, ActQuant2, ProjSt2)            new stock is inserted
        else  +stock_of_products(Part, ActQuant, ProjStock)),       else new stock is created
              -product_order(P, _, _, _, _,_, _, _, _, _, _)        product order is deleted
  else  if(stock_of_products(Part, ActQuant, ProjStock)           else production order, if stock exists, then
        then addstock(ProjStock, (P, Quant, Time), ProjSt2),          production is added to projected stocks
              -stock_of_products(Part, _, _),                         old stock is deleted
              +stock_of_products(Part, ActQuant, ProjSt2)             new stock is inserted
        else  +stock_of_products(Part, 0, [(P, Quant, Time)])),     else new stock is created
              -product_order(P, _, _, _, _,_, _, _, _, _, _),       old product order is deleted
              +product_order(P, 3, Part, Time, Quant, FromAct,      product order marked as cancelled is inserted
                  FromProj, Output, Price, Customer, ProdList)).

retstock([Entry|Rest], Stock, Stock2) ←                           old stocks and returned stocks are merged
  retstock(Rest, Stock, Stock3),                                  recursive call of rule
  addstock(Stock3, Entry, Stock2).                                appends one returned stock to old stock list
retstock([ ], Stock, Stock) .                                     exit rule of recursion

addstock(Stock, (P, Quant, Time), Stock2) ←                       adds stock to stock list
  if(lmember((P, Quant2, Time), Stock)                            if entry with same order and time exists,
  then remstock(Stock, (P, Quant2, Time), Stock3),                then it is deleted from the list
    Quant3=Quant+Quant2,                                            returned quantity is added to old one
    insstock(Stock3, (P, Quant3, Time), Stock2)                     and new entry is inserted
  else insstock(Stock, (P, Quant, Time), Stock2)).                else stock is inserted as new entry

remstock(Stock, Entry, Stock2) ←                                  removes entry from stock list
  Stock=[(P, Quant, Time)|Rest],                                  retrieves first entry from list
  Entry=(P2, _, _),                                               retrieves product order from new entry
  if(P=P2                                                         if product orders are equal
  then Stock2=Rest                                                  entry is deleted
  else remstock(Rest, Entry, Rest2),                              recursive call of rule
    Stock2=[(P, Quant, Time)|Rest2]).                               first entry is inserted again
remstock([ ], _, [ ]).                                            exit rule of recursion

insstock(Stock, Entry, Stock2) ←                                  inserts entry into stock list
  Stock=[(P, Quant, Time)|Rest],                                  retrieves first entry from list
  Entry=(_, _, Time2),                                            retrieves time of new entry
  if(Time2 <= Time                                                if new time is earlier or equal
  then Stock2=[Entry|Rest]                                          new stock is inserted
  else  insstock(Rest, Entry, Rest2),                             recursive call of rule
        Stock2=[(P, Quant, Time)|Rest2]).                           first entry is inserted again
insstock([ ], Entry, [Entry]).                                    exit rule of recursion
```

order data (Q9):

q9($Order_number, Result)

```
q9(Order_number, Result) ←
  if(Order_number=all                                 if no selection for specific order number
  then qu_order_all(Result)                           then all orders are retrieved
  else qu_order_single(Order_number, Result)).        else single order is retrieved
qu_order_single(O, (P, Part, Time, Quant, Price)) ←   retrieval of single order
  order(O, P, Part, Time, Quant, Price).
qu_order_all(<(O, P, Part, Time, Quant, Price)>) ←    retrieval of set of all orders
  order(O, P, Part, Time, Quant, Price).
```

master data of standard parts by supplier (Q20):

q20($Supplier, Result)

```
q20(Supplier, Result) ←
  if(Supplier=all                                     if no selection for specific supplier
  then qu_supplier_all(Result)                        then all standard parts are retrieved
  else qu_supplier_single(Supplier, Result)).         else standard parts for single supplier are retrieved
qu_supplier_all(<(Supplier, Entry)>) ←                retrieval of all standard parts grouped by supplier
  supplier(_, Supplier, _),                           retrieving all suppliers
  qu_supplier_single(Supplier, Entry).                retrieving all standard parts for single supplier
qu_supplier_single(Supplier,                          retrieving set of standard parts for single supplier
    (<Name, Standard, Price, Min_Quant, Time)>) ←
  if(standard_part(N2, S2, P2, Supplier, M2, T2),     if standard parts exist
  then  Name=N2,                                      then they are included as set members
        Standard=S2,
        Price=P2,
        Min_Quant=M2,
        Time=T2
  else  Name=none,                                    else a dummy member is created
        Standard=none,
        Price=0.0,
        Min_Quant=0,
        Time=0).
```

assignment of actions to operators and vice versa (Q35):

q35($Criterion, $Direction, Result)

```
q35(Criterion, Direction, Result) ←
  if(Direction=1                                      if assignment of actions to operators
  then  if(Criterion=all                              then, if no selection for specific operator
        then qu_actop_all(Result)                       then all actions are retrieved
        else qu_actop_single(Criterion, Result))        else actions for single operator are retrieved
  else  if(Criterion=all                              else assigning operators to actions, if no selection
        then qu_opact_all(Result)                       then all operators are retrieved
        else qu_opact_single(Criterion, Result))).      else operators for single action are retrieved
qu_actop_all(<(Operator, Entry)>) ←                   retrieval of all actions grouped by operator
  operator(_, Operator, _, _, _, _, _, _),            retrieving all operators
  qu_actop_single(Operator, Entry).                   retrieving all actions for single operator
qu_actop_single(Operator, (<Action>) ←                retrieving set of actions for single operator
  operator(_, Operator, _, _, _, Qualification, _, _), retrieving qualification of operator
  if(member(Machine_type, Qualification),             retrieving machine types in qualification
    action(Action2, Machine_type, _, _, _, _)         if actions with corresponding machine type exist
  then Action=Action2                                 then they are included as set members
  else Action=none).                                  else a dummy member is created
```

(qu_opact_all and qu_opact_single are analogous to qu_actop_all and qu_actop_single)

inventory of assembled parts (Q46):

```
q46($Assembled_part, Result)

q46(Assembled_part, Result) ←
  assembled_part(Assembled_part, _),               checks for existence of assembled part
  hierarchy(Assembled_part, Hierarchy),            sub-part hierarchy is computed
  inv(Hierarchy, List_of_AssParts, 1),             computes inventory of assembled parts
  inv2(Hierarchy, List_of_BasicParts),             computes inventory of basic parts
  Result={(List_of_AssParts, List_of_BasicParts)}.

hierarchy(Part, List2) ←                           recursive computation of sub-part hierarchy
  assembled_part(Part, List),                      if assembled part then
  take_apart(List, List2).                         computes hierarchy for entries in sub-part list
hierarchy(Part, [ ]) ←                             exit rule of recursion
  basic_part(Part, _, _, _, _).

take_apart([(Quant, Part)|Rest],                   computes hierarchy of entries in sub-part list
    [(Quant, Part, List)|Rest2]) ←
  hierarchy(Part, List),                           for each sub-part compute sub-part hierarchy
  take_apart(Rest, Rest2).                         recursive call of rule
take_apart([ ], [ ]).                              exit rule of recursion

inv([Entry|Rest], List, I) ←                       computes inventory of assembled parts
  Rest ~= [ ],                                     checks that list has more than one entry
  inv([Entry], List1, I),                          recursive call for list entry
  inv(Rest, List2, I),                             recursive call for rest of list
  merge(List1, List2, List).                       merging of both partial results
inv([(Quant, Part, List)], List2, I) ←             rule for single entry
  List ~= [ ],                                     checks that list is not empty
  J = I + 1,                                       increases level of hierarchy (=search depth)
  inv(List, List3, J),                             recursive call of rule
  factor(Quant, List3, List4),                     multiply list members by required quantity
  assembled_part(Part, List5),                     retrieving sub-part list of entry
  compress(List5, List6),                          removing quantity information from sub-part list
  List1 = [(Quant, Part, I, List6)],               inventory entry: quantity, part, level, sub-part names
  merge(List1, List4, List2).                      merging of new entry with other result
inv([(_, _, [ ])], [ ], _).                        exit rule for basic parts
inv([ ], [ ], _).                                  exit rule for empty list

factor(Quant, [(Quant1, Part, Level, Subparts)|Rest],   multiplies sub-part list by required quantity
    [(Quant2, Part, Level, Subparts)|Rest2]) ←
  Quant2=Quant1*Quant,
  factor(Quant, Rest, Rest2).                      recursive call of rule
factor(_, [ ], [ ]).                               exit rule of recursion

merge([(Quant, Part, Level, Subparts)|Rest], List, List2) ←   merging of two inventories
  insert(Quant, Part, Level, Subparts, List, List3),   inserts first entry of first list in second list
  merge(Rest, List3, List2).                       recursive call
merge(List, [ ], List).                            exit rule of recursion
merge([ ], List, List).                            exit rule of recursion

insert(Quant1, Part1, Level1, Subparts1,           inserts inventory entry in second inventory
    [(Quant2, Part2, Level2, Subparts2)|Rest],
    [(Quant, Part2, Level, Subparts2)|Rest2]) ←
  if(Part1=Part2                                   if first entry of list belongs to the same part
  then Quant=Quant1+Quant2,                        then add up quantities
    Rest2=Rest ,
    max(Level1, Level2, Level)                     takes level at which part is needed first
  else insert(Quant1, Part1, Level1, Subpart1, Rest, Rest2),   else recursive call of rule
    Quant=Quant2,
    Level=Level2).
insert(Quant, Part, Level, Subparts, [ ],          if no entry for part exists, a new entry is created
    [(Quant, Part, Level, Subparts)]).
```

(inv2 is analogous to inv)

7.5 Semantic Application Model

The final step of the development of the functional part was to specify *semantic categories* for all manipulations and queries to the PPC (see Figure 33 and Figure 35 for the corresponding LDL base predicates). The resulting semantic application model of the PPC provided a well-defined interface to the natural language front-end.

In the following we define for each semantic category its deep structure and specify the valid combinations of value domains of the applied arguments. In addition to the three basic data types, the types address and name with the appropriate syntactic restrictions are used. For arguments of which the value domain is derived from the existent PPC data, the corresponding type information is written in italics. Finally, the use of underline means that the concerned words do not represent data types but have to be regarded literally.

7.5.1 Manipulations to the PPC

Insertion of new entities:

↳ [insert, Entity_type, Entity, [Values]] (7.1)

No.	Entity_type	Entity	[Values]
M1	supplier	name	[address]
M2	raw_material	string	[real, *supplier,* integer, integer]
M3	standard_part	string	[string, real, *supplier,* integer, integer]
M4	cast_part	string	[string, real, *supplier,* integer, integer]
M5	round_part	string	[*raw_material,* integer, integer]
M6	angular_part	string	[*raw_material,* integer, integer, integer]
M7	assembled_part	string	[(integer,*part*)], [(integer,*part*)*,* (integer,*part*)], ...
M8	product	string	[integer, real, real]
M9	customer	name	[address]
M10	machine	none	[string]
M11	operator	name	[string, address, real, {string}]
M12	action	string	[*machine_type,* real, real, real, real]
M13	op_sequence	*part*	[*action*], [*action, action*], ...

Table 1: Value domains of semantic category *insert*

Deletion of an entity:

↳ [delete, Entity_type, Entity] (7.2)

No.	Entity_type	Entity
M14	supplier	*supplier*
M15	basic_part	*basic_part*
M16	raw_material	*raw_material*
M17	assembled_part	*assembled_part*
M18	product	*product*
M19	machine	integer
M20	operator	*operator*
M21	customer	*customer*
M22	action	*action*
M39	stock_of_basic_parts	*basic_part*
M40	stock_of_products	*product*

Table 2: Value domains of semantic category *delete*

Update of an attribute:

↳ [update, Entity_type, Attribute, Entity, Value] (7.3)

No.	Entity_type	Entity	Attribute	Value
M23	material	*material*	price	real
M24	material	*material*	quantity	integer
M25	material	*material*	time	integer
M26	material	*material*	supplier	*supplier*
M27	supplier	*supplier*	address	address
M28	product	*product*	quantity	integer
M29	product	*product*	profit_margin	real
M30	product	*product*	cancellation_fee	real
M31	vat_rate	none	vat_rate	real
M32	operator	*operator*	address	address
M33	operator	*operator*	salary	real
M34	operator	*operator*	machine_type	string
M35	customer	*customer*	address	address
M36	action	*action*	fixed	(real, real)
M37	action	*action*	variable	(real, real)

Table 3: Value domains of semantic category *update*

Shift of time:

↳ [timeshift, Days, Hours] (7.4)

No.	Days	Hours
M38	integer	integer

Table 4: Value domains of semantic category *timeshift*

Machine stoppages and operator drop outs:

↳ [failure, Entity_type, Entity, Days, Hours] (7.5)

No.	Entity_type	Entity	Days	Hours
M43	machine	integer	integer	integer
M44	operator	*operator*	integer	integer

Table 5: Value domains of semantic category *failure*

Release of machines and operators:

↳ [release, Entity_type, Entity] (7.6)

No.	Entity_type	Entity
M46	machine	integer
M47	operator	*operator*

Table 6: Value domains of semantic category *release*

Calculation of new selling prices:

↳ [calcprice] (7.7)

M41

Suspension of machines and operators:

➭ [suspend, Entity_type, Entity] (7.8)

No.	Entity_type	Entity
M48	machine	integer
M49	operator	*operator*

Table 7: Value domains of semantic category *suspend*

Scheduling of product order:

➭ [scheduling, Product, Quantity, Customer] (7.9)

No.	Product	Quantity	Customer
M42	*product*	integer	*customer*

Table 8: Value domains of semantic category *scheduling*

Delivery delays:

➭ [delay, Order, Days] (7.10)

No.	Order	Days
M45	integer	integer

Table 9: Value domains of semantic category *delay*

Cancellation of product order:

➭ [cancel, Product_order] (7.11)

No.	Product_order
M50	integer

Table 10: Value domains of semantic category *cancel*

7.5.2 Queries to the PPC

All queries are mapped to one homogenous semantic category:

➭ [query, Entity_type, Attribute, Entity, Sel_entity_type, Sel_entity] (7.12)

Table 11 shows the values for all queries. If a list of all entities shall be retrieved, Entity is set to all. The argument Sel_entity refers to the entity which is used as selection criterion. If no selection is required, then Sel_entity_type=Sel_entity=none, for the grouping of the query result the value all has to be assigned to Sel_entity (none or all are only entered in the table if no other choice exists).

According to the semantics of assignments (Q33-Q37), the selection criterion is always required. Finally, the values status and type_of_stock of Sel_entity_type can only be applied as grouping operators.

No.	Entity_type	Entity	Attribute	Sel_entity_type	Sel_entity
Q1	ass_part	*assembled_part*	subparts	none	none
Q45	ass_part	*assembled_part*	sphierarchy	none	none
Q46	ass_part	*assembled_part*	inventory	none	none
Q36	ass_part	*assembled_part*	assignment	supplier	*supplier*
Q37	ass_part	*assembled_part*	assignment	machine_type	*machine_type*
Q2, Q21	standart_part	*standard_part*	masterdata	supplier	*supplier*
Q3, Q22	cast_part	*cast_part*	masterdata	supplier	*supplier*
Q4, Q23	round_part	*round_part*	masterdata	raw_material	*raw_material*
Q5, Q24	angular_part	*angular_part*	masterdata	raw_material	*raw_material*
Q6, Q25	raw_material	*raw_material*	masterdata	supplier	*supplier*
Q7	supplier	*supplier*	masterdata	none	none
Q8, Q38	basic part	*basic part*	stock	type_of_stock	all
Q19, Q32	basic part	*basic part*	masterdata	supplier	*supplier*
Q9, Q25	order	integer	orderdata	product_order	integer
Q26	order	integer	orderdata	basic_part	*basic_part*
Q33	order	integer	assignment	product_order	integer
Q10, Q27	prod_order	integer	prorderdata	product	*product*
Q28	prod_order	integer	prorderdata	customer	*customer*
Q40	prod_order	integer	prorderdata	status	all
Q33	prod_order	integer	assignment	order	integer
Q11, Q41	machine	integer	machdata	status	all
Q29	machine	integer	machdata	machine_type	*machine_type*
Q12, Q42	operator	*operator*	assigndata	status	all
Q30	operator	*operator*	assigndata	machine_type	*machine_type*
Q13	operator	*operator*	masterdata	none	none
Q35	operator	*operator*	assignment	action	*action*
Q14, Q31	action	*action*	masterdata	machine_type	*machine_type*
Q35	action	*action*	assignment	operator	*operator*
Q15	part	*part*	opsequence	none	none
Q47	part	*part*	prodlist	none	none
Q50	part	*part*	included	direct, indirect	asspart, product
Q49	part	*part*	costdistr	quantity	integer
Q16	product	*product*	masterdata	none	none
Q17, Q39	product	*product*	stock	status	all
Q43	product	*product*	prodtime	quantity	integer
Q48	product	*product*	pricediff	none	none
Q34	product	*product*	assignment	customer	*customer*
Q36	product	*product*	assignment	supplier	*supplier*
Q37	product	*product*	assignment	machine_type	*machine_type*
Q44	time_interval	integer	events	hours	integer
Q37	mach_type	*machine_type*	assignment	product	*product*
Q37	mach_type	*machine_type*	assignment	assembledpart	*assembled_part*

Table 11: Value domains of semantic category *query*

7.6 Empirical Collection of Test Data

The final part of the design and implementation of our PPC database system was the development of the natural language interface. As in our opinion an inadequate interface represents the main obstacle for the broad user acceptance of any database application, the careful elaboration of this task embodies particular significance. This is especially true for both the field of deductive databases [Lockemann92] and natural language interfaces [Bates87].

For this reason we did not invent any artificial queries or manipulations but applied questionnaires to obtain realistic input sentences. Based on this empirical data we implemented in *Section 7.7 Implementation of the Natural Language Interface* the interface by integrating the developed concepts and tools and adapting them to the specific requirements of the application. Finally, we optimised the prototype system with regard to efficiency and transparency and evaluated its feasibility in *Section 7.8 Evaluation* by extensive test cycles.

For the creation of an appropriate test data collection we designed two different types of questionnaires. Questionnaire A was addressed to persons with a good knowledge of relational database algebra. We transformed the 50 manipulations and 50 queries to equivalent pseudo-code constructs of a relational database query language (SQL) for which the interviewed person should find a corresponding natural language input sentence. Figure 36 shows as example the entry for M35 (update of address for customers, kunde=customer, anschrift=address, the input sentence would read in English: *Mr. Anton Huber has moved to 1220 Wien, Lieblgasse 53.*)

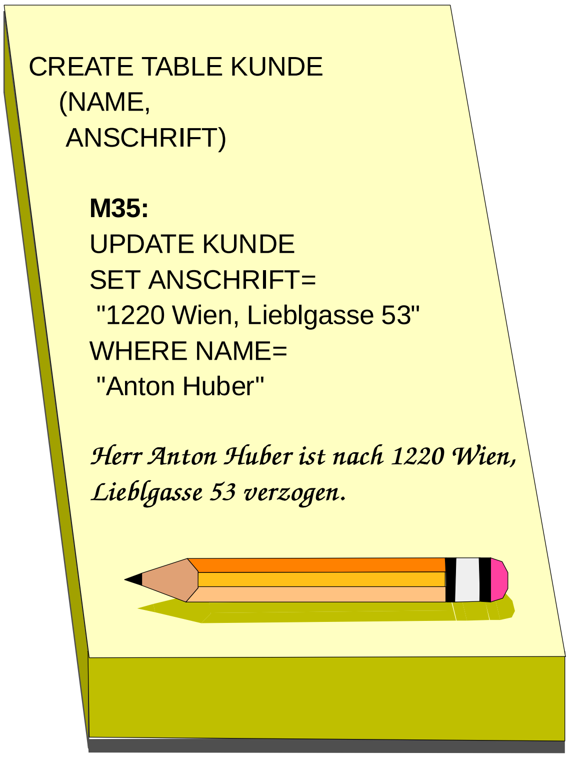

Figure 36: Example of questionnaire A

The particular advantage of using a formal language was that the person could not adjust his answers to given natural language phrases. Therefore, the capability of free word association was not restricted in any way.

Since we regarded the restriction of asking only persons with sound background in computer science as severe limitation in order to achieve practice-oriented query patterns, we also included other people in our sample. As for persons without the knowledge of SQL this kind of questionnaire was not applicable, we designed a second type, questionnaire B, which confronted the user with the command prototypes (see *Section 7.4.3 Implementation*) and an explanation of the applied arguments. Then we stated an example command by use of the internal query form and asked for a corresponding natural language expression. Again, in Figure 37 the pattern for M35 can be seen (the English translation of the user input: *New address of Mr. Anton Huber is 1220 Wien, Lieblgasse 53.*)

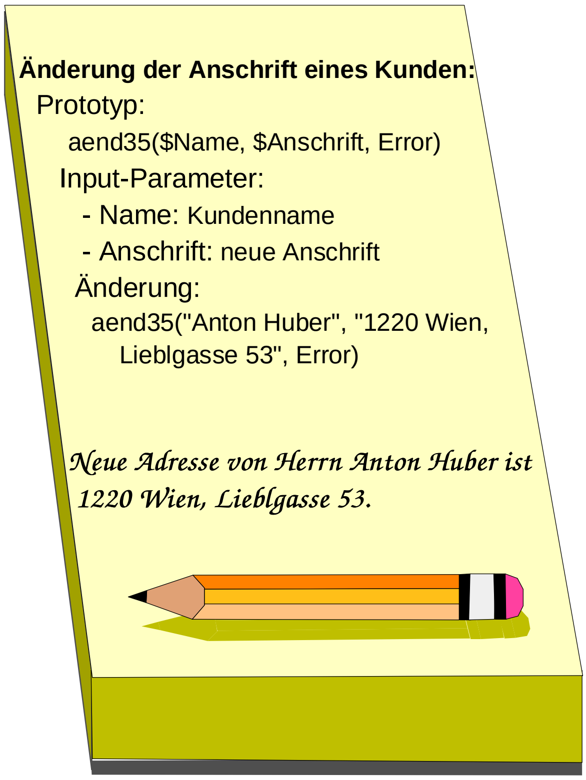

Figure 37: Example of questionnaire B

For each type of questionnaire we performed five interviews so that we resulted in a test collection of 1000 natural language sentences (see the appendix for two original examples). Besides the broad coverage of linguistic phenomena this large quantity of data was especially necessary to verify the selectivity of semantic analysis, that is, the correct mapping of each 10 different surface structures for the same command to one specific semantic deep structure.

7.7 Implementation of Natural Language Interface

The first step of implementation was to construct the dictionary as explained in *Section 4.5 Implementation.* Table 12 shows the final number of entries for each category. The small total amount of 431 entries which were necessary to cover all 1000 input sentences illustrates once more the compact storage structure resulting from the application of the IDA architecture.

Word category	Quantity
adjective	32
adjectival suffix	6
adverb	28
article	12
pronoun	33
conjunction	7
numeral	14
preposition	27
substantive	78
substantival suffix	9
verb	119
verb prefix	8
verb form	58

Table 12: Number of dictionary entries for PPC

Whereas for the morphological analysis only minor adaptations to the already developed tools were necessary, of course for the implementation of the semantic analysis component more work had to be done in order to establish a homogenous transition to the semantic application model of the PPC. With regard to syntactic analysis we applied the UVL-analysis method (see *Section 6.2 Semantic Analysis*), that is, we did not produce complete grammatical structures of input sentences but based the semantic analysis directly on the deep form list produced by morphological analysis using syntactic knowledge only if necessary for disambiguation. This choice was made possible due to the careful design process of the PPC which resulted in a well-defined semantic application model, therefore making the semantic analysis a rather straight-forward and natural task.

Figure 38 shows part of the LDL code that performs the semantic analysis of the queries declared in *Section 7.4.2 Queries to the PPC.* First, the semantic category query is selected according to the entries in the deep form list (DFL), the unknown value list (UVL), and the unknown type list (UTL). Then the correct mapping of the entity types and entities for the query and the optional selection or grouping criterion is determined. This is performed on the one hand by analysing the deep forms included in the DFL, on the other hand by applying *mapping patterns* according to the entries in Table 11 which are modelled by use of the LDL base predicate qusemtype. This makes it possible to adapt the semantic analysis to new query types in a flexible way by simply updating the LDL database without any change to the rule base. The schema of qusemtype is defined as follows:

qusemtype(Sem: {string}, Etype: string, Attr: string, SelEtype: string) (7.13)

Sem refers to the deep forms included in DFL, e.g.:

Q2: ({standard_part}, standard_part, masterdata, none)
Q21: ({standard_part, supplier}, standard_part, masterdata, supplier)

semanalyse(DFL, UVL, UTL, Result) ←	semantic analysis based on UVL analysis
search_cat(DFL, UVL, UTL, Cat, Etype, Entity),	determines semantic category
semant(Cat, DFL, UVL, UTL, Etype, Entity, Result).	semantic analysis for semantic category
search_cat(DFL, [(Entity, [string | _])], [Etype], query,	queries where a specific entity is given
Etype, Entity) ←	aside from machines, orders, and product orders
Etype~=unknown,	entity must be valid entity type
~sem_cat(DFL, _).	no other semantic category applies
search_cat(DFL, [(Entity, [integer])], _, query,	queries for specific machines, orders,
integer, Entity) ←	and product orders
~sem_cat(DFL, _).	no other semantic category applies
search_cat(DFL, _, [ ], query, unknown, unknown) ←	queries for list of all entities
~sem_cat(DFL, calcprice).	semantic category calcprice does not apply
semant(query, DFL, _, _, Etype, Entity, Result) ←	semantic analysis of queries
if(Entity~=unknown	if query for specific entity or selection criterion
then qusemsel(DFL, Etype, Entity, Result)	then appropriate semantic analysis
else qusemall(DFL, Result)).	else query for list of all entities
qusemsel(DFL, Etype, Entity, Result) ←	query for specific entity aside from machines, ...
qusem2(DFL, Sem),	retrieving deep forms from DFL
union(Sem, {Etype}, Sem2),	joining deep forms with entity type
qusemsel2(Sem2, Entity, Result).	generating deep structure of query
qusemsel(DFL, integer, Entity, Result) ←	query for specific entity for machines, ...
qusem2(DFL, Sem),	retrieving deep forms from DFL
qusemsel2(Sem, Entity, Result).	generating deep structure of query
qusemsel2(Sem, Entity, Result) ←	generating deep structure of query
qusemtype(Sem, Etype, Attr, SelEtype),	retrieving mapping pattern
if(SelEtype=none	if no selection
then Entity2=Entity,	then specific entity is retrieved
SelEntity=none	
else Entity2=all,	else entity is used as selection criterion
SelEntity=Entity),	
Result=[query, Etype, Attr, Entity2, SelEtype, SelEntity].	resulting deep structure
qusemall(DFL, Result) ←	queries for list of all entities
qusem(DFL, Sem),	retrieving deep forms from DFL
qusemtype(Sem, Etype, Attr, SelEtype),	retrieving mapping pattern
if(SelEtype=none	decides if grouping is applied
then SelEntity=none	
else SelEntity=all),	
Result=[query, Etype, Attr, all, SelEtype, SelEntity].	resulting deep structure
qusem2(DFL, Sem) ←	retrieving deep forms from DFL
if(qusem(DFL, Sem2)	if deep forms are included in DFL
then Sem=Sem2	then they are returned
else Sem={ }).	else empty set is returned
qusem(DFL, <Sem>) ←	grouping of individual solutions
qusemant(DFL, Sem).	retrieving deep forms from DFL

Figure 38: LDL code of semantic analysis for PPC

If semantic analysis does not result in a unique interpretation, the following reasons are probable:

- ☹ the input sentence is in contradiction with the semantic application model
- ☹ the input sentence contains spelling errors
- ☹ relevant information is missing

Whereas there is no possibility to correct the first situation, the two other faults can be possibly corrected by applying spelling error correction and pragmatic analysis. With regard to spelling error correction, we applied the algorithm presented in *Section 6.4 Spelling Error Correction* for the correction of misspelled database values (as threshold value we have chosen *+0,5*). For pragmatic analysis (see *Section 6.3. Pragmatic Analysis*) we applied the proposed uniform semantic resolution method (USRM) of using the entity type and the entity of the precedent command as antecedent for discourse resolution.

Spelling error correction and pragmatic analysis are only applied if the prior semantic analysis does not produce a unique deep structure. Thus, we resulted in a modified process model shown in Figure 39 different from the standard model in Figure 2.

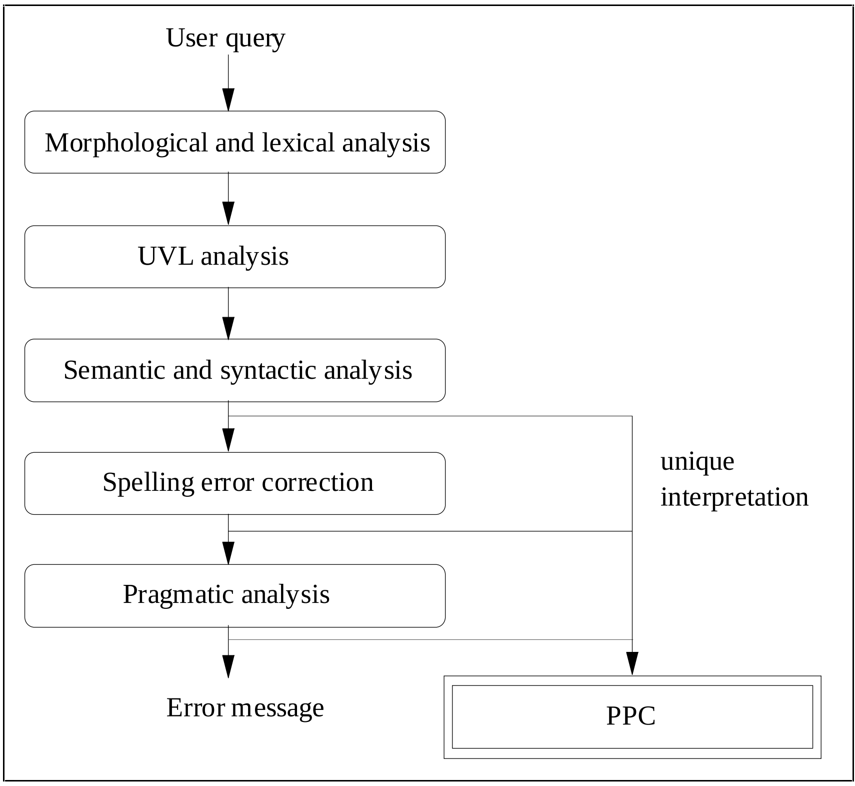

Figure 39: Process model of natural language analysis in IDA

For incorrect sentences which still cannot be analysed correctly an appropriate error message is created. Three different types of erroneous output of analysis might occur:

- ☹ no solution
- ☹ a solution with unknown arguments
- ☹ several solutions

7.8 Evaluation

The main task of the final evaluation step was to verify the faultless mapping of the 1000 input sentences to the 100 commands of the PPC database system. After extensive testing cycles all natural language input was correctly analysed. As second step, the correct functionality of the spelling error correction and pragmatic analysis modules was checked and proven flawless. In the following we give some examples of the evaluation study also including cases of misspelled input and missing information.

Example 7.1:

Lösche Kurbel (=delete handle) M17

Morphological and lexical analysis (see Figure 19) results in:

DFL: [{(loesch, verb, [loesch])}, {('Kurbel', unknown, [string])}]

UVL analysis (see Figure 21) results in:

USL: [[('Kurbel', [string])]]
UVL: [('Kurbel', [string])]
UTL: [assembled_part]

Semantic analysis (see Figure 38) results in:

SDS: {[delete, assembled_part, 'Kurbel']}

Since semantic analysis produces a unique interpretation, no syntactic analysis, spelling error correction or pragmatic analysis is needed and the final response of the system is:

Bauteil noch in Bauteilen enthalten, darf nicht gelöscht werden
(=part is still included in other parts, must not be deleted) ■

Example 7.2:

Neue Tätigkeit: Polieren Parameter sind Drehbank, Fixzeit: 10,0 Fixkosten: 45 Variabler Zeitanteil: 2,4 Variable Kosten 15,2
(=new action: to polish parameters are lathe, fixed time: 10,0 fixed cost 45, variable time portion: 2,4 variable cost 15,2) M12

USL: [[('Polieren', [string])], [('Drehbank', [string])], [('10,0', [real])], [('45', [integer])],[('2,4', [real])], [('15,2', [real])]]
UVL: [('Polieren', [string]), ('Drehbank', [string]), (10.0, [real]), (45, [integer]), (2.4, [real]), (15.2, [real])]
UTL: [unknown, machine_type, unknown, unknown, unknown, unknown]

SDS: {[insert, action, 'Polieren', ['Drehbank', 10.0, 45.0, 2.4, 15.2]]}

Änderung erfolgreich durchgeführt.
(=update successfully performed) ■

Example 7.3:

⌨ *Aktualisiere variablen Zeitanteil auf 2.7 und Kosten auf 15.9.*
(=update variable time portion to 2.7 and cost to 15.9.) M37

USL: [[('2.7', [real])], [('15.9', [real])]]
UVL: [(2.7, [real]), (15.9, [real])]
UTL: [unknown, unknown]

Since it is not stated which entity has to be updated, no unique semantic analysis can be obtained. In context with *Example 7.2* and by applying pragmatic analysis (see Figure 24) the following de-referencing can be performed:

Entity_type: action
Entity: Polieren

SDS: {[update, action, fixed, 'Polieren', (2.7, 15.9)]}

💻 Änderung erfolgreich durchgeführt.
(=update successfully performed) ■

Example 7.4:

⌨ *Zeige die Stammdaten für Normteile von Egon Müler*
(=show the master data for standard parts of Egon Müler) Q20

UVL: [[('Egon', [string]), ('Mueler', [string])]]
USL: [('Egon Mueler', [string, string])]
UTL: [unknown]

Because of the typing error in the surname of the supplier, the entity in question cannot be retrieved from the PPC system. Therefore, the module for the spelling error correction (see Figure 29) is applied in order to find the correct spelling:

USL: [('Egon Mueller', [string, string])]
UTL: [supplier]

SDS: {[query, masterdata, standard_part, all, supplier, 'Egon Mueller']}

💻

Lieferant	Normteil	Norm	Kosten	Minmenge	Lieferzeit
Egon Müller	Kegelstift 3x30	DIN 1	2.70	50	7
	Zylinderschraube M 8x15	ÖNORM M5119 5.6	1.50	100	7

■

Besides the faultless operation, the basic requirement for the feasibility of the practical use of any database application is its *performance*. The main measure that has to be tested in this context is of course the *response time*. We performed careful tests and measuring, the results are shown in Table 13, the mean response time for each command category is given in seconds and hundredths of seconds. Furthermore, the results are divided in the response time of the interface, the database system, and the total response time.

Commands	Interface	Database	Total
M1-M13	5:29	5:19	10:48
M14-M22	2:14	5:04	7:18
M23-M37	4:19	5:44	9:63
M38-M50	2:56	10:91	13:47
Q1-Q19	3:01	0:09	3:10
Q20-Q32	3:33	0:12	3:45
Q33-Q37	3:89	0:10	3:99
Q38-Q50	3:47	6:93	10:40

Table 13: Response times of PPC

The overall mean response time for all queries was 7:71 (3:48 for interface and 4:23 for the database system (as hardware configuration we used a SUN SPARC 10 station). These satisfactory performance results also only slightly increase if one includes spelling error correction and pragmatic analysis. Table 14 compares the results (total response times) for the third group of commands.

	Default	Spelling errors	Pragmatic analysis
Response Time	4:19	5:89	5:33
Difference		1:70	1:14

Table 14: Response times of additional features

7.9 Summary

In this final Section we have applied the methods and tools of the Integrated Deductive Approach to a practice-oriented case study, the development of a PPC for a precision tool factory. For this purpose we proposed the following *seven step model*:

① requirements analysis
② database definition
③ specification and implementation of the functionality
④ semantic application model
⑤ empirical collection of test data
⑥ implementation of natural language interface
⑦ evaluation

The careful elaboration of each of these steps forms the solid basis for a successful database application. As concerns the natural language part, we identified step 5 to possess particular significance because it guarantees the complete coverage of all occurring linguistic phenomena and therefore wide user acceptance for later use in practice.

8. Résumé

We have identified in our research the following main characteristics of natural language database interfaces in contrast to other fields of natural language processing: specific application domains with well-defined semantics, rather small delimited vocabularies, mappings to simple target representations, short input sentences without complex linguistic phenomena but including misspellings, ungrammatical or incomplete statements.

The main reason why many previous attempts to build successful natural language interfaces failed can be seen in the fact that those characteristics were neglected. The use of sophisticated techniques that maybe worked very well for other applications are simply oversized for database interfaces, therefore obstructing the way to efficient solutions. In this context also the popular term 'domain-independent' must be regarded with critical reservation. Many authors claim to build domain-independent interfaces by ignoring the available application-specific data. As we have pointed out, only a domain-dependent interface can operate efficiently by making full use of the information which can be derived from the underlying database system. This is not necessarily in contradiction with portability because also such systems can be designed and implemented in view of later easy portation to other application areas. Even if some previous work came to the same conclusions, the limitations of relational database technology represented an obstacle too high to overcome. Only with the emergence of deductive database technology there exists for the first time a computational framework that combines the required operational power with a purely declarative semantics leading the way to clear and concise realisations of natural language interfaces.

We see the main contribution of this thesis in the introduction of a new kind of architecture, the *Integrated Deductive Approach* to efficient natural language interfaces which regards the interface in contrast to other existent work not as loosely coupled filter but as integral part of the database system itself. By the use of the powerful logic language provided by deductive databases we guarantee a homogenous mapping of the input to the corresponding database commands over all steps of analysis. Although all concepts and tools in this work have been developed for German, they incorporate the capacity to be applied also to other languages, especially to inflexional and free word order languages. We have proven the feasibility of our approach by an extensive case study for which we proposed a *seven step methodology*, its central point is the *empirical collection of test data* in order to guarantee complete *customisation* for later practical use. Further research in this topic will include portability studies to other applications and languages as well as investigations on the adaptive behaviour of natural language interfaces, e.g. the consideration of new functional words or changes to the application model. We believe that the ideas proposed in this thesis represent a challenging application of deductive databases as well as contribute an important step forward to the development of efficient natural language interfaces with widespread user acceptance.

Acknowledgement

The author is grateful to J. Eder for the many helpful hints on deductive database technology and to the 10 interviewees to carry out the tedious task of data collection. Without their enthusiasm the implementation of the presented case study would have been impossible. Special thanks are due to my supervisors A M. Tjoa and G. Vinek for their valuable advice during the completion of this thesis.

References

[Ackley90a] D. Ackley et.al.; Systems Analysis for Deductive Database Environments: an Enhanced Role for Aggregate Entities; Proc. 9th Int. Conf. Entity-Relationship Approach; 1990

[Ackley90b] D. Ackley et.al.; Process-Object-State Modelling, a Proprietary, Multi-Dimensional Modeling Method; Ackley Associates, Fremont; 1990

[Aho72] A.V. Aho, J.D. Ullman; The Theory of Parsing, Translation and Compiling; Prentice Hall, Englewood Cliffs; 1972

[Ali86] Y. Ali, R. Aubin, B. Hall; A Domain-Independent Natural Language Database Interface; Proc. Canadian Conf. Artificial Intelligence; 1986

[Ali93] S.S. Ali; A Propositional Semantic Network with Structured Variables for Natural Language Processing; Proc. 6th Australian Joint Conf. Artificial Intelligence; 1993

[Allen80] J.F. Allen, C.R. Perrault; Analyzing Intentions in Utterances; Artificial Intelligence; Vol. 15, No. 3; 1980

[Allen87] J.F. Allen; Natural Language Understanding; Benjamin/Cummings, Menlo Park; 1987

[Antonacci89] F. Antonacci et.al.; A System for Text Analysis and Lexical Knowledge Acquisition; Data & Knowledge Eng.; Vol. 4; 1989

[Aoe90] J. Aoe; A Method for Building Knowledge Bases with Morphological Semantics; SIGIR Forum; Vol. 24, No. 1-2; 1990

[Appelt93] D.E. Appelt et.al.; FASTUS: A Finite-state Processor for Information Extraction from Real-world Text; Proc. Int. Joint Conf. Artificial Intelligence; 1993

[Arens93] Y. Arens, E. Hovy, S. van Mulken; Structure and Rules in Automated Multimedia Presentation Planning; Proc. Int. Joint Conf. Artificial Intelligence; 1993

[Baetge84] J. Baetge et.al.; Vahlens Kompendium der Betriebswirtschaftslehre (in German); Vahlen, München; 1984

[Bancilhon85] F. Bancilhon; Naive Evaluation of Recursively Defined Relations; M.L. Brodie, J. Mylopoulos (eds.); Springer, New York; 1985

[Bancilhon86a] F. Bancilhon et.al.; Magic Sets and Other Strange Ways to Implement Logic Programs; Proc. ACM SIGMOD-SIGACT Symp. Principles Database Systems; 1986

[Bancilhon86b] F. Bancilhon, R. Ramakrishnan; An Amateur's Introduction to Recursive Query Processing; Proc. ACM-SIGMOD Conf.; 1986

[Bar-Hillel64] Y. Bar-Hillel; On Categorial and Phrase Structure Grammars; in: Y. Bar-Hillel (ed.); Language and Information; Addison-Wesley, Reading; 1964

[Bates78] M. Bates; The Theory and Practice of Augmented Transition Networks; in: L. Bloc (ed.); Natural Language Communication with Computers; Springer, Berlin; 1978

[Bates87] M. Bates; Natural-Language Interfaces; in: S.C. Shapiro, D. Eckroth (eds.); Encyclopaedia of Artificial Intelligence; Wiley, New York; 1987

[Baudin93] C. Baudin, J.G. Underwood, V. Baya; Using Device Models to Facilitate the Retrieval of Multimedia Design Information; Proc. Int. Joint Conf. Artificial Intelligence; 1993

[Bear88] J. Bear; Generation and Recognition of Inflectional Morphology; Proc. Österreichische Artificial-Intelligence-Tagung; 1988

[Beeri87a] C. Beeri, R. Ramakrishnan; On the Power of Magic; Proc. ACM SIGMOD-SIGACT Symp. Principles Database Systems; 1987

[Beeri87b] C. Beeri et.al.; Sets and Negation in a Logical Database Language (LDL); Proc. ACM SIGMOD-SIGACT Symp. Principles Database Systems; 1987

[Beeri88] C. Beeri; Data Models and Languages for Databases; Proc. 2nd Int. Conf. Database Theory; 1988

[Beeri89] C. Beeri et.al.; Set Constructors in a Logic Database Language; J. Logic Programming; 1989

[Beler93] M. de Beler, X. Huang, C. Rowles; Meaning in Spoken English; Proc. Workshop Natural Language Processing, 6th Australian Joint Conf. Artificial Intelligence; 1993

[Belew87] R.K. Belew; A Connectionist Approach to Conceptual Information Retrieval; Proc. Int. Conf. Artificial Intelligence & Law; 1987

[Bench-Capon93] T. Bench-Capon; Neural Network and Open Texture; Proc. Int. Conf. Artificial Intelligence & Law; 1993

[Berry-Rogghe78] G.L. Berry-Rogghe, H. Wulz; An Overview of PLIDIS: A Problem Solving Information System with German as a Query Language; in: L. Bolc (ed.); Natural Language Communication with Computers; Springer, Berlin; 1978

[Bertino92] E. Bertino, B. Catania, G. Guerrini; Towards a Logical-Object Oriented Programming Language for Databases; Proc. Int. Conf. Extending Database Technology; 1992

[Bertino93] E. Bertino, B. Catania, G. Guerrini; An Overview of LOL: a Deductive Language for Object Bases; Proc. Int. Symp. Next Generation Database Systems and Their Applications; 1993

[Berwick84] R.C. Berwick, R.C. Weinberg; The Grammatical Basis of Linguistic Performance: Language Use and Acquisition; MIT Press, Cambridge, MA.; 1984

[Bickel87] M.A. Bickel; Automatic Correction to Misspelled Names: a Fourth- Generation Language Approach; CACM; Vol. 30, No. 3; 1987

[Binot84] J.-L. Binot; A Set-Oriented Semantic Network Formalism for the Representation of Sentence Meaning; Proc. European Conf. Artificial Intelligence; 1984

[Binot86] J.-L. Binot, D. Ribbens; Dual Frames: A New Tool for Semantic Parsing; Proc. Conf. of the American Association for Artificial Intelligence; 1986

[Birnbaum81] L. Birnbaum, M. Selfridge; Conceptual Analysis of Natural Language; in: R. Schank, C. Riesbeck (eds.); Inside Computer Understanding; Lawrence Erlbaum, Hillsdale; 1981

[Black93] A. Black; Using Situation Theory in a Computational Language for Natural Language Processing; Proc. Fourth Int. Workshop Natural Language Understanding and Logic Programming; 1993

[Bobrow80] R.J. Bobrow, B.L. Webber; Knowledge Representation for Syntactic/Semantic Processing; Proc. Conf. of the American Association for Artificial Intelligence; 1980

[Bocca86] J. Bocca; On the Evaluation Strategy of Educe; Proc. ACM-SIGMOD Conf. Management of Data; 1986

[Bollinger89] T. Bollinger, U. Hedtstück, C.-R. Rollinger; Reasoning for Text Understanding - Knowledge Processing in the 1st LILOG-Prototype; Proc. German Workshop Artificial Intelligence; 1989

[Boral88] H. Boral; Parallelism in Bubba; Proc. Int. Symp. Databases Parallel Distributed Systems; 1988

[Bosc86] P. Bosc, M. Courant, S. Robin; CALIN: A User Interface Based on a Simple Natural Language; Proc. ACM Conf. in Information Retrieval; 1986

[Bouma88] G. Bouma; Modifiers and Specifiers in Categorial Unification Grammar; Computational Linguistics; Vol. 26; 1988

[Bracchi83] G. Bracchi, S. Ceri, G. Pelagatti; A Set of Integrated Tools for the Conceptual Design of Database Schemas and Transactions; in: S. Ceri (ed.); Methodology and Tools for Data Base Design; North-Holland, Amsterdam; 1983

[Bras90] M. Bras; Representing Discursive Temporal Knowledge: A Computational Application of DRT; in: S. Rhamani, R. Chandrasekar, K.S.R. Anjaneyulu (eds.); Knowledge Based Computer Systems; Springer, Berlin; 1990

[Brewka93] G. Brewka, K. Konolige; An Abductive Framework for General Logic Programs and Other Nonmonotonic Systems; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Briscoe93] T. Briscoe, J. Carroll; Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars; Computational Linguistics; Vol. 19, No. 1; 1993

[Brown75] J.S. Brown, R.R. Burton; Multiple Representations of Knowledge for Tutorial Reasoning; in: D.G. Bobrow, A. Collins (eds.); Representation and Understanding; Academic Press, New York; 1975

[Brown90] P.F. Brown et.al.; A Statistical Approach to Machine Translation; Computational Linguistics; Vol. 16, No. 2; 1990

[Brown93] P.F. Brown et.al.; The Mathematics of Statistical Machine Translation: Parameter Estimation; Computational Linguistics; Vol. 19, No. 2; 1993

[Cacace89] F. Cacace et.al.; Integrating Object-Oriented Data Modelling with a Rule-Based Programming Paradigm; Proc. ACM Int. Conf. Management of Data; 1989

[Carberry89] S. Carberry; A Pragmatics-Based Approach to Ellipsis Resolution; Computational Linguistics; Vol. 15, No. 2; 1989

[Capindale90] R.A. Capindale, R.G. Crawford; Using a Natural Language Interface with Casual Users; Int. J. Man-Machine Studies; Vol. 32; 1990

[Cappelli84] A. Cappelli et.al.; A Framework for Integrating Syntax and Semantics; in: B.G. Bara, G. Guida (eds.); Computational Models of Natural Language Processing; North-Holland, Amsterdam; 1984

[Caraceni93] R. Caraceni et.al.; Integrating Data and Text Retrieval in a Natural Language System; Proc. 13th Int. Conf. Artificial Intelligence, Expert Systems and Natural Language; 1993

[Carpenter91] B. Carpenter; The Generative Power of Categorial Grammars and Head-Driven Phrase Structure Grammars with Lexical Rules; Computational Linguistics; Vol. 17, No. 3; 1991

[Castelfranchi84] C. Castelfranchi, D. Parisi, O. Stock; Knowledge Presentation and Natural Language: Extending the Expressive Power of Proposition Nodes; in: B.G. Bara, G. Guida (eds.); Computational Models of Natural Language Processing; North-Holland, Amsterdam; 1984

[Ceri86] S. Ceri, G. Gottlob, L. Lavazza; Translation and Optimization of Logic Queries: the Algebraic Approach; Proc. 12th Int. Conf. Very Large Data Bases; 1986

[Ceri87a] S. Ceri, L. Tanca; Optimization of Systems of Algebraic Equations for Evaluating Datalog Queries; Proc. 13th Int. Conf. Very Large Data Bases; 1987

[Ceri87b] S. Ceri, G. Gottlob, G. Wiederhold; Interfacing Relational Databases and Prolog Efficiently; in: L. Kerschberg (ed.); Expert Database Systems; Benjamin/Cummings, Menlo Park; 1987

[Ceri88] S. Ceri et.al.; The ALGRES Project; Proc. Int. Conf. Extending Database Technology; 1988

[Ceri89] S. Ceri, G. Gottlob, L. Tanca; What You Always Wanted to Know About Datalog (And Never Dared to Ask); IEEE Trans. on Knowledge and Data Engineering; Vol. 1, No. 1; 1989

[Ceri90] S. Ceri, G. Gottlob, L. Tanca; Logic Programming and Databases; Springer, Berlin; 1990

[Chandra85] A. Chandra, D. Harel; Horn Clause Queries and Generalizations; J. Logic Programming; Vol. 1; 1985

[Chang81] C.L. Chang; On the Evaluation of Queries Containing Derived Relations in Relational Databases; in: H. Gallaire, J. Minker, J.M. Nicolas (eds.); Advances in Database Theory, Vol. 1; Plenum, New York; 1981

[Chang86] C.L. Chang, A. Walker; PROSQL: A Prolog Programming Interface with SQL/DS; in: L. Kerschberg (ed.); Expert Database Systems; Benjamin/Cummings, Menlo Park; 1986

[Charniak86] E. Charniak, D. McDermott; Introduction to Artificial Intelligence; Addison-Wesley, Reading; 1986

[Chen76] P. Chen; The Entity-Relationship Model: Towards a Unified View of Data; ACM Trans. on Database Systems; Vol. 1, No. 1; 1976

[Chiaramella87] Y. Chiaramella, B. Defude; A Prototype of an Intelligent System for Information Retrieval: IOTA; Information Processing & Management; Vol. 23, No. 4; 1987

[Chimenti87] D. Chimenti et.al.; An Overview of the LDL System; IEEE Data Engineering, Special Issue on Database and Logic; Vol. 10, No. 4; 1987

[Chimenti89a] D. Chimenti, R. Gamboa, R. Krishnamurthy; Towards An Open Architecture in LDL; Proc. 15th Conf. Very Large Data Bases; 1989

[Chimenti89b] D. Chimenti, R. Gamboa, R. Krishnamurthy; Using Modules and Externals in LDL; Tech. Rep., ACA-ST-268-89, MCC; 1989

[Chimenti89c] D. Chimenti, R. Gamboa; The SALAD Cookbook: A User's Guide; Tech. Rep., ACA-ST-064-89, MCC; 1989

[Chimenti89d] D. Chimenti, R. Gamboa, R. Krishnamurthy; Abstract Machine for LDL; Tech. Rep., ACA-ST-268-89, MCC; 1989

[Chimenti90] D. Chimenti et.al.; The LDL System Prototype; IEEE Trans. on Knowledge and Data Eng.; Vol. 2, No. 1; 1990

[Chinchor92] N. Chinchor; MUC-4 Evaluation Metrics; Proc. Fourth Message Understanding Conf.; 1992

[Chinchor93] N. Chinchor, L. Hirschman, D.D. Lewis; Evaluating Message Understanding Systems: An Analysis of the Third Message Understanding Conference (MUC-3); Computational Linguistics; Vol. 19, No. 3; 1993

[Chomsky56] N. Chomsky; Three Models for the Description of Language; IRE Trans. PGIT; Vol. 2; 1956

[Chu93] J. Chu; Responding to User Queries in a Collaborative Environment; Proc. 31st Annual Meeting of the Association for Computational Linguistics; 1993

[Church93] K.W. Church, R.L. Mercer; Introduction to the Special Issue on Computational Linguistics Using Large Corpora; Computational Linguistics; Vol. 19, No. 1; 1993

[Codd74] E.F. Codd; Seven Steps to RENDEZVOUS with the Casual User; IBM Research Rep., J1333; San Jose Research Laboratory; 1974

[Cohen78] P.R. Cohen; On Knowing what to Say: Planning Speech Acts; Tech. Rep., TR 188; Univ. Toronto; 1978

[Cohen79] P.R. Cohen, C.R. Perrault; Elements of a Plan-Based Theory of Speech-Acts; Cognitive Science; Vol. 3; 1979

[Cohen85] P.R. Cohen, H.J. Levesque; Speech Acts and Rationality; Proc. 23rd Annual Meeting of the Association for Computational Linguistics; 1985

[Colmerauer78] A. Colmerauer; Metamorphosis Grammars; in: L. Bloc (ed.); Natural Language Communication with Computers; Springer, Berlin; 1978

[Copestake90] A. Copestake, K.S. Sparck-Jones; Natural Language Interfaces to Databases; Knowledge Engineering Review; Vol. 5, No. 5; 1990

[Croft87] W.B. Croft, R.H. Thompson; I^3R: A New Approach to the Design of Document Retrieval Systems; JASIS; Vol. 38, No. 6; 1987

[Cruttenden86] A. Cruttenden; Intonation; Cambridge Univ. Press, Cambridge; 1986

[Cuppens86] F. Cuppens, R. Demolombe; A PROLOG-Relational DBMS Interface Using Delayed Evaluation; Proc. Workshop Integration of Logic Programming and Databases; 1986

[Czejdo88] B. Czejdo, C.F. Eick, M. Taylor; Integrating Sets, Rules, and Data in an Object-Oriented Environment; IEEE Expert; 1988

[Daelemans92] W. Daelemans, A. van den Bosch; A Neural Network for Hyphenation; Proc. Int. Conf. Artificial Neural Networks; 1992

[Damerau81] F. Damerau; Operating Statistics for the Transformational Question Answering System; AJCL; Vol. 7, No. 1; 1981

[Danforth85] S. Danforth, S. Khoshafian, P. Valduriez; FAD - A Database Programming Language, Rev. 2; Tech. Rep., ACA-DB-151-85, MCC; 1988

[Dillon83] M. Dillon, L.K. McDonald; Fully Automatic Book Indexing; J. Documentation; Vol. 39, No. 3; 1983

[Dorffner85] G. Dorffner, H. Trost; Morphologische Analyse und intelligente Fehlerkorrektur in VIE-LANG (in German); Proc. Österreichische Artificial-Intelligence-Tagung; 1985

[Dorr90] B.J. Dorr; Machine Translation: A Principle-Based Approach; in: P.H. Winston, S.A. Shellard (eds.); Artificial Intelligence at MIT. Expanding Frontiers; MIT Press, Cambridge, MA.; 1990

[Doszkocs86] T.E. Doszkocs; Natural Language Processing in Information Retrieval; JASIS; Vol. 37, No. 4; 1986

[Dowding93] J. Dowding et.al.; Gemini: a Natural Language System for Spoken-Language Understanding; Proc. 31st Meeting of the Association for Computational Linguistics; 1993

[Dowty87] D. Dowty; Type Raising, Functional Composition, and Non-Constituent Conjunction; in: R. Oehrle, E. Bach, D. Wheeler (eds.); Categorial Grammar and Natural Language Structures; Reidel, Dordrecht; 1987

[Dunin-Keplicz84] B. Dunin-Keplicz; Default Reasoning in Anaphora Resolution; Proc. European Conf. Artificial Intelligence; 1984

[Earley70] J. Earley; An Efficient Context-Free Parsing Algorithm; CACM; Vol. 13, No. 2; 1970

[Elmasri85] R. Elmasri, A. Hevner, J. Weeldreyer; The Category Concept: An Extension to the Entity-Relationship Model; Data Knowledge Eng.; Vol. 1, No. 1; 1985

[El-Sharkawi90] M. El-Sharkawi, Y. Kambayashi; The Architecture and Implementation of ENLI: Example-Based Natural Language-Assisted Interface; Proc. Int. Conf. Databases, Parallel Architectures and Their Applications; 1990

[Emele88] M. Emele; Überlegungen zu einer Two-level Morphologie (in German); Proc. Österreichische Artificial-Intelligence-Tagung; 1988

[Evans87] R. Evans; Direct Interpretation of the GPSG Formalism; in: J. Hallam, C. Mellish (eds.); Advances in Artificial Intelligence; Wiley, Chinchester; 1987

[Fass83] D. Fass, Y. Wilks; Preference Semantics, Ill-Formedness, and Metaphor; Computational Linguistics; Vol. 9, No. 3-4; 1983

[Fillmore68] C.J. Fillmore; The Case for Case; in: E. Bach, R. Harms (eds.); Universals in Linguistic Theory; Holt, Rinehart, and Winston, New York; 1968

[Fillmore77] C.J. Fillmore; The Case for Case Reopened; in: P. Cole, J. Sadock (eds.); Syntax and Semantics: Grammatical Relations; Academic Press, New York; 1977

[Finkler88] W. Finkler, G. Neumann; MORPHIX: A Fast Realization of a Classification-Based Approach to Morphology; Proc. Österreichische Artificial-Intelligence-Tagung; Springer, Berlin; 1988

[Fisher89] A.J. Fisher; Practical Parsing of Generalized Phrase Structure Grammars; in: Computational Linguistics; Vol. 15, No. 3; 1989

[Flickinger85] D. Flickinger, C. Pollard, T. Wasow; Structure-sharing in Lexical Representation; Proc. Annual Meeting of the Association for Computational Linguistics; 1985

[Fliegner88] M. Fliegner; HOKUSKOPUS - Verwendung räumlichen Wissens bei der Analyse von Quantorenskopus und Distributivität (in German); Proc. German Workshop Artificial Intelligence; 1988

[Frederking88a] R.E. Frederking; Integrated Natural Language Dialogue; Kluwer, Boston; 1988

[Frederking88b] R.E. Frederking, M. Gehrke; Resolving Anaphoric References in a DRT-based Dialogue System; Proc. 4. Österreichische Artificial-Intelligence-Tagung; 1988

[Frederking93] R.E. Frederking et.al.; The Integration of MT and MAT; Proc. First Conf. of the Pacific Association for Computational Linguistics; 1993

[Fritsch85] A.M. Fritsch; Using Model Theory to Specify AI Programs; Proc. Int. Joint Conf. Artificial Intelligence; 1985

[Gal85] A. Gal, J. Minker; A Natural Language Database Interface that Provides Cooperative Answers; in: C.R. Weisbin (ed.); Artificial Intelligence Applications; IEEE Press, Washington; 1985

[Gallaire84] H. Gallaire, J. Minker, J.M. Nicolas; Logic and Databases: a Deductive Approach; Computing Surveys; Vol. 16, No. 2; 1984

[Gazdar82] G. Gazdar; Phrase Structure Grammar; in: P. Jacobson, G.K. Pullum (eds.); The Nature of Syntactic Representation; Reidel, Dordrecht; 1982

[Gazdar85] G. Gazdar et.al.; Generalized Phrase Structure Grammar; Blackwell, Oxford; 1985

[Gazdar89] G. Gazdar, C. Mellish; Natural Language Processing in POP-11; Addison-Wesley, Wokingham; 1989

[Gebruers88] R. Gebruers; Valency and MT. Recent Developments in the METAL System; 26th Annual Meeting of the Association for Computation Linguistics; 1988

[Genikomsidis88] D. Genikomsidis; Eine französische Two-Level-Morphologie (in German); Stuttgart University Press, Stuttgart; 1988

[Ghose93] A.K. Ghose et.al.; Iterated Belief Change: A Preliminary Report; Proc. 6th Australian Joint Conf. Artificial Intelligence; 1993

[Greco90] S. Greco, N. Leone, P. Rullo; COMPLEX: An Object-Oriented Logic Programming System; IEEE Trans. on Knowledge and Data Eng.; Vol. 4, No. 4; 1990

[Grosz82] B.J. Grosz; Transportable Natural-Language Interface. Problems and Techniques; Proc. 20th Annual Meeting of the Association for Computational Linguistics; 1982

[Grosz83] B.J. Grosz; Team: A Transportable Natural-Language Interface System; Proc. Conf. Applied Natural Language Processing; 1983

[Grosz86] B.J. Grosz, C. Sidner; Attention, Intention, and the Structure of Discourse; Computational Linguistics; Vol. 12, No. 3; 1986

[Haller93] S.M. Haller; An Interactive Model for Plan Explanation; Proc. 6th Australian Joint Conf. Artificial Intelligence; 1993

[Han93] Y. Han, I. Zukerman; Towards a Planning Mechanism for Multimedia Presentation; Proc. Workshop Natural Language Processing, 6th Australian Joint Conf. Artificial Intelligence; 1993

[Harris84] L.R. Harris; Natural Front Ends; in: P.H. Winston, K.A. Prendergast (eds.); The AI Business; MIT Press, Cambridge, MA.; 1984

[Hauenschild88] C. Hauenschild; GPSG and German Word Order; in: U. Reyle, C. Rohrer (eds.); Natural Language Parsing and Linguistic Theories; Reidel, Dordrecht; 1988

[Helbig86] H. Helbig; Syntactic-Semantic Analysis of Natural Language by a New Word-Class Controlled Functional Analysis (WCFA); Computers and AI; Vol. 5, No. 1; 1986

[Hendrix78] G.G. Hendrix et.al.; Developing a Natural Language Interface to Complex Data; ACM Trans. on Database Systems; Vol. 3, No. 2; 1978

[Henschen84] L.J. Henschen, S.A. Naqvi; On Compiling Queries in Recursive First Order Databases; J. ACM; Vol. 31, No. 1; 1984

[Hintikka69] J. Hintikka; Semantics for Propositional Attitudes; in: J.W. Davis, D.J. Hockney, K.W. Wilson (eds.); Philosophical Logic; Reidel, Dordrecht; 1969

[Hirst81] G. Hirst; Anaphora in Natural Language Understanding; Springer, Berlin; 1981

[Hirst82] G. Hirst, E. Charniak; Word Sense and Case Slot Disambiguation; Conf. of the American Association for Artificial Intelligence; 1982

[Hobbs87] J.R. Hobbs, S.M. Shieber; An Algorithm for Generating Quantifier Scopings; Computational Linguistics; Vol. 13; 1987

[Hobbs88] J.R. Hobbs et.al.; Interpretation as Abduction; Proc. 26th Annual Meeting of the Association for Computational Linguistics; 1988

[Hobbs92] J.R. Hobbs et.al.; SRI International: Description of the FASTUS System Used for MUC-4; Proc. Fourth Message Understanding Conf.; 1992

[Hoeksema84] J. Hoeksema; Categorial Morphology; Diss. Univ. Groningen; 1984

[Höfferer94a] M. Höfferer, B. Knaus, W. Winiwarter; Using Genetics in Information Filtering; Proc. 22nd Annual Conf. of the Canadian Association for Information Science; 1994

[Höfferer94b] M. Höfferer, B. Knaus, W. Winiwarter; Cognitive Filtering of Information by Genetic Algorithms; Proc. Workshop Genetic Algorithms within the Framework of Evolutionary Computation, 18. Deutsche Jahrestagung für Künstliche Intelligenz; 1994

[Höfferer94c] M. Höfferer, B. Knaus, W. Winiwarter; Adaptive Information Filtering by Monitoring User Behavior; Proc. 8th Int. Symp. Methodologies for Intelligent Systems; 1994

[Höfferer94d] M. Höfferer, B. Knaus, W. Winiwarter; Adaptive Information Extraction from Online Messages; Proc. RIAO, Intelligent Multimedia Information Systems and Management; 1994

[Hoffman93] B. Hoffman; The Formal Consequences of Using Variables in CCG Categories; Proc. 31st Annual Meeting of the Association for Computational Linguistics; 1993

[Höppner83] W. Höppner et.al.; Beyond Domain Independence: Experience with the Development of a German Language Access System to Highly Diverse Background Systems; Proc. Int. Joint Conf. Artificial Intelligence; 1983

[Hovy91] E.H. Hovy; Approaches to the Planning of Coherent Text; in: C.L. Paris, W.R. Swartout, W.C. Mann (eds.); Natural Language Generation in Artificial Intelligence and Computational Linguistics; Kluwer, Boston; 1991

[Hui88] L.L. Hui, I. Zukerman; Common-sense Resolution of Syntactic Ambiguity in Database Queries; Proc. 1st Australian Joint Conf. Artificial Intelligence; 1988

[Hui93] S.C. Hui, A. Goh, J.K. Raphel; CLOG: A Class-Based Logic Language for Object-Oriented Databases; Int. Symp. Object Technologies for Advanced Software; 1993

[Hui94] S.C. Hui, A. Goh, J.K. Raphel; The CLOGBase Programming Environment for Constructing Intelligent Database Applications; Proc. Int. Conf. Data and Knowledge Systems for Manufacturing and Engineering; 1994

[Hull92] J.J. Hull, Combining Syntactic Knowledge and Visual Text Recognition: a Hidden Markov Model for Part of Speech Tagging in a Word Recognition Algorithm; Proc. AAAI Symp. Probabilistic Approaches to Natural Language Processing; 1992

[Hsu93] L.S. Hsu, C.L. Tan, Z. Wu; Handling Real World Input by Abduction; Proc. 1st Conf. of the Pacific Association for Computational Linguistics; 1993

[Ioannidis87] Y.E. Ioannidis et.al.; BERMUDA - An Architectural Perspective on Interfacing Prolog to a Database Machine; Tech. Rep. 723, Univ. Wisconsin; 1987

[Isermann79] H. Isermann; Strukturierung von Entscheidungsprozessen bei mehrfacher Zielsetzung (in German); OR Spektrum; Vol. 1, No. 3; 1979

[Ishikawa93] I. Ishikawa et.al.; The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System; ACM Trans. on Database Systems; Vol. 18, No. 1; 1993

[Jacobs90] P.S. Jacobs, L.F. Rau; SCISOR: Extracting Information from On-line News; CACM; Vol. 33, No. 4; 1990

[Jacquemin93] C. Jaquemin; A Coincidence Detection Network for Spatio-Temporal Coding: Application to Nominal Composition; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Jain91] A.N. Jain, A.H. Waibel; Parsing with Connectionist Networks; in: M. Tomita (ed.); Current Issues in Parsing Technology; Kluwer, Boston; 1991

[Jajodia91] S. Jajodia, R.S. Sandhu; Toward a Multilevel Secure Relational Data Model; Proc. ACM Int. Conf. Management of Data (SIGMOD); 1991

[Jarke86] M. Jarke, J. Clifford, M. Vassiliou; An Optimizing Prolog Front End to a Relational Query System; Proc. ACM - SIGMOD Conf. Management of Data; 1986

[Jensen87] K. Jensen, J. Binot; Disambiguating Prepositional Phrase Attachments by Using On-line Dictionary Definitions; Computational Linguistics; Vol. 13, No. 3-4; 1987

[Kalita86] J.K. Kalita, M.A. Jones, G.I. McGalla; Summarizing Natural Database Responses; Computational Linguistics; Vol. 12, No. 2; 1986

[Kambayashi86] Y. Kambayashi; An Overview of a Natural Language-Assisted Database User Interface: ENLI; Proc. IFIP 10th World Computer Congress; 1986

[Kamp81] H. Kamp; A Theory of Truth and Semantic Representation; in: J.A.G. Groendijk, T.M.V. Jansen, M.B.J. Stokhof (eds.); Formal Methods in the Study of Language; Mathematical Centre Tracts, Amsterdam; 1981

[Kamp88] H. Kamp; Discourse Resolution Theory: What it is and where it Ought to Go; in: A. Blaser (ed.); Natural Language at the Computer; Springer, Berlin; 1988

[Kaplan73] R.M. Kaplan; A General Syntactic Processor; in: R. Rustin (ed.); Natural Language Processing; Algorithmics Press, New York; 1973

[Kaplan82] R.M. Kaplan, J. Bresnan; Lexical-Functional Grammar: A Formal System for Grammatical Representation; in: J. Bresnan (ed.); The Mental Representation of Grammatical Relations; MIT Press, Cambridge, MA; 1982

[Kappel94] G. Kappel, S. Vieweg; Database Requirements for CIM Applications; J. Integrated Manufacturing; 1994

[Karttunen87] L. Karttunen et.al.; TWOL: A Compiler for Two-Level Phonological Rules; in: Dalrymple et.al. (eds.); Tools for Morphological Analysis; Stanford University Press, Stanford; 1987

[Katz88] B. Katz; Text Processing with the START Natural Language System; in: E. Barrett (ed.); Text, ConText, and HyperText; MIT Press, Cambridge, MA.; 1988

[Katz89] E. Katz et.al.; A Mathematical Model for Translation of Natural Languages; Information Sciences; Vol. 47; 1989

[Kay73] M. Kay; The MIND System; in: R. Rustin (ed.); Natural Language Processing; Algorithmics Press, New York; 1973

[Kay85] M. Kay; Parsing in Functional Unification Grammar; in: D.R. Dowty, L. Karttunen, A. Zwicky (eds.); Natural Language Parsing; Cambridge Univ. Press, Cambridge, MA.; 1985

[Keenan91] F.G. Keenan, L.J. Evett, R.J. Withdraw; A Large Vocabulary Stochastic Syntax Analysis for Handwriting Recognition; Proc. First Int. Conf. Document Analysis; 1991

[Kehler93] A. Kehler; The Effect of Establishing Coherence in Ellipsis and Anaphora Resolution; Proc. 31st Annual Meeting of the Association for Computational Linguistics; 1993

[Kifer86] M. Kifer, E.L. Lozinskii; Filtering Data Flow in Deductive Databases; Proc. 1st Int. Conf. Database Theory; 1986

[Kitano93] H. Kitano; A Comprehensive and Practical Model of Memory-Based Machine Translation; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Kittredge86] B. Kittredge, A. Polguère, E. Goldberg; Natural Language Report Synthesis: An Application to Marine Weather Forecasts; Proc. Canadian Conf. Artificial Intelligence; 1986

[König89] E. König; Parsing as Natural Deduction; 27th Annual Meeting of the Association for Computational Linguistics; 1989

[Koskenniemi83] K. Koskenniemi; Two-level Morphology: A General Computational Model for Wordform Recognition and Production; Helsinki University Press, Helsinki; 1983

[Kripke63] S. Kripke; Semantical Consideration on Modal Logic; Acta Philosophica Fennica; Vol. 16; 1963

[Krishnamurthy88a] R. Krishnamurthy, S. Naqvi; Non-Deterministic Choice in Datalog; Proc. 3rd Int. Conf. Data Knowledge Bases; 1988

[Krishnamurthy88b] R. Krishnamurthy, C. Zaniolo; Optimization in a Logic Based Language for Knowledge and Data Intensive Applications; Proc. Int. Conf. Extending Database Technology; 1988

[Kunifji84] S. Kunifji, H. Yokota; Prolog and Relational Databases for the 5th Generation Computer Systems; in: H. Gallaire, J. Minker, J.M. Nicolas (eds.); Advances in Logic and Databases, Vol. 2; Plenum Press, New York; 1984

[Küpper82] H.-U. Küpper; Ablauforganisation (in German); Fischer, Stuttgart; 1982

[Lassez88] J.-L. Lassez, M.J. Maher, K. Marriott; Unification Revisited; in: J. Minker (ed.); Foundations of Deductive and Logic Programming; Kaufmann, Los Altos; 1988

[Lee90] K. Lee, S. Lee; An Object-Oriented Approach to Data/Knowledge Modelling Based on Logic; Proc. IEEE Conf. Data Engineering; 1990

[Lefebre89] A. Lefebre, L. Vielle; On Deductive Query Evaluation in the DedGin System; Proc. 1st Int. Conf. Deductive and O-O Databases; 1989

[Levesque84] H.J. Levesque; A Logic of Implicit and Explicit Belief; Proc. Conf. of the American Association for Artificial Intelligence; 1984

[Lewis89] D.D. Lewis, W.B. Croft, N. Bhandaru; Language-Oriented Information Retrieval; Int. J. Intelligent Systems; Vol. 4; 1989

[Lewis92] D.D. Lewis, R.M. Tong; Text Filtering in MUC-3 and MUC-4; Proc. 4th Message Understanding Conf.; 1992

[Li84] D. Li; A Prolog Database System; Research Institute Press, Letchworth; 1984

[Litman87] D.J. Litman, J.F. Allen; A Plan Recognition Model for Subdialogues in Conversations; Cognitive Science; Vol. 11, No. 2; 1987

[Lloyd87] J.W. Lloyd; Foundations of Logic Programming; Springer, New York; 1987

[Lockemann92] P.C. Lockemann; Object-Oriented Databases and Deductive Databases: Systems Without Markets ? Market Without Systems ?; Proc. Int. Conf. Database and Expert Systems Applications; 1992

[Lou91] Y. Lou, Z.M. Ozsoyoglu; LLO: An Object-Oriented Deductive Language with Methods and Methods Inheritance; Proc. ACM Int. Conf. Management of Data; 1991

[Luckhardt82] H.-D. Luckhardt; Keine Sorge - «SUZY» geht noch in den Kindergarten. Zum derzeitigen Stand der maschinellen Sprachübersetzung (MÜ) am Beispiel des Saarbrücker Übersetzungsmodells «SUZY» (in German); Deutscher Drucker; Vol. 3; 1982

[Luger89] G.F. Luger, W.A. Stubblefield; Artificial Intelligence and the Design of Expert Systems; Benjamin/Cummings, Redwood City; 1989

[Lunin84] L. Lunin, L. Smith; Perspectives on Artificial Intelligence: Concepts, Techniques, Applications, Promise; JASIS; Vol. 25, No. 4; 1984

[Lytinen84] S.L. Lytinen, A. Gershman; ATRANS: Automatic Processing of Money Transfer Messages; Proc. Conf. of the American Association for Artificial Intelligence; 1984

[Lytinen86] S.L. Lytinen; Dynamically Combining Syntax and Semantics in Natural Language Processing; Proc. Conf. of the American Association for Artificial Intelligence; 1986

[Magerman94] D.M. Magerman; Natural Language Parsing as Statistical Pattern Recognition; Diss., Stanford Univ.; 1994

[Maier88] P. Maier, P. Steffens; Determinatoren und Quantoren in einer kategorialen Unifikationsgrammatik des Deutschen (in German); Proc. 4. Österreichische Artificial-Intelligence Tagung; 1988

[Makuta-Giluk93] M. Makuta-Giluk, C. DiMarco; A Computational Formalism for Syntactic Aspects of Rhetoric; Proc. 1st Conf. of the Pacific Association for Computational Linguistics; 1993

[Matsumoto93] Y. Matsumoto, Y. Den, K. Yanagi; A Flexible Natural Language System with Concurrency and Meta-level Processing; Proc. Fourth Int. Workshop Natural Language Understanding and Logic Programming; 1993

[McCabe92] F. McCabe; Logic and Objects; Prentice Hall, Englewood Cliffs; 1992

[McCarthy80] J. McCarthy; Circumscription: A Form of Non-Monotonic Reasoning; Artificial Intelligence; Vol. 13; 1980

[McCord85] M.C. McCord; Modular Logic Grammars; Proc. Annual Meeting of the Association for Computational Linguistics; 1985

[McDonald87] D.D. McDonald; Natural-Language Generation; in: S.C. Shapiro, D. Eckroth (eds.); Encyclopedia of Artificial Intelligence; Wiley, New York; 1987

[McFetridge88] P. McFetridge et.al.; System X: A Portable Natural Language Interface; Proc. 7th Biennial Conf. of the Canadian Society for Computational Studies of Intelligence; 1988

[McFetridge90] P. McFetridge, C. Groeneboer; Novel Terms and Coordination in a Natural Language Interface; in: S. Ramani, R. Chandrasekar, K.S.R. Anjaneyulu (eds.); Knowledge Based Computer Systems; Springer, Berlin; 1990

[McKeown93] K.R. McKeown; Language Generation for Multimedia Explanations; Proc. First Conf. of the Pacific Association for Computational Linguistics; 1993

[Meknavin93] S. Meknavin, T. Theeramunkong, H. Tanaka; Parsing Ill-Formed Input with ID/LP Rules; Proc. 4th Int. Workshop Natural Language Understanding and Logic Programming; 1993

[Merkl94a] D. Merkl, E. Schweighofer, W. Winiwarter; CONCAT - Connotation Analysis of Thesauri Based on the Interpretation of Context Meaning; Proc. 5th Int. Conf. Database and Expert Systems Applications; 1994

[Merkl94b] D. Merkl, E. Schweighofer, W. Winiwarter; Automatic Knowledge Acquisition for Creating Nuclear Juridical Lexica; Proc. Post-COLING Int. Workshop Directions of Lexical Research; 1994

[Millies89] S. Millies; Kategoriales Parsing mit definiten Klauseln (in German); Proc. 13. Jahrestagung Künstliche Intelligenz; 1989

[Minsky75] M. Minsky; A Framework for Representing Knowledge; in: P. Winston (ed.); The Psychology of Computer Vision; McGraw-Hill, New York; 1975

[Mittal93] V.O. Mittal, C.L. Paris; Automatic Documentation Generation: The Interaction of Text and Examples; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Moore73] R.C. Moore; D-SCRIPT: a Computational Theory of Description; Proc. Int. Joint Conf. Artificial Intelligence; 1973

[Moore85] R.C. Moore; Semantical Considerations on Nonmonotonic Logic; Artificial Intelligence; Vol. 25; 1985

[Moore91] J.D. Moore, W.R. Swartout; A Reactive Approach to Explanation: Taking the User's Feedback into Account; in: C.L. Paris, W.R. Swartout, W.C. Mann (eds.); Natural Language Generation in Artificial Intelligence and Computational Linguistics; Kluwer, Boston; 1991

[Morris86] K. Morris, J.D. Ullman, A. Van Gelder; Design Overview of the Nail! System; Proc. Int. Conf. Logic Programming; 1986

[Morris87] K. Morris et.al.; YAWN! (Yet Another Window on NAIL!); IEEE Data Eng., Special Issue on Databases and Logic; Vol. 10, No. 4; 1987

[Mundt88] B. Mundt; Fehlerlokalisation und Korrektur von Textwörtern mit Trigrammlexika (in German); Sprache und Datenverarbeitung; Vol. 1; 1988

[Nagao93a] K. Nagao; Abduction and Dynamic Preference in Plan-Based Dialogue Understanding; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Nagao93b] K. Nagao, K. Hasida, T. Miyata; Understanding Spoken Natural Language with Omni-Directional Information Flow; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Naqvi87] S. Naqvi; A Logic for Negation in Database Systems; in: J. Minker (ed.); Foundations of Deductive Databases and Logic Programming; Kaufmann, Los Altos; 1987

[Naqvi88] S. Naqvi, R. Krishnamurthy; Semantics of Updates in Logic Programming; Proc. 7th ACM SIGMOD-SIGACT Symp. Principles Database Systems; 1988

[Naqvi89] S. Naqvi, S. Tsur; A Logical Language for Data and Knowledge Bases; Computer Science Press, Rockville; 1989

[Navathe83] S. Navathe, A. Cheng; A Methodology for Database Schema Mapping from Extended Entity Relationship Models into the Hierarchical Model; in: G.C. Davis et.al. (eds.); The Entity-Relationship Approach to Software Engineering; Elsevier, New York; 1983

[Nijholt88] A. Nijholt; Computers and Languages; North-Holland, Amsterdam; 1988

[Novak89] H.-J. Novak, B. Wesche; Analyse und Synthese in einer kategorialen Unifikations-grammatik: Möglichkeiten und Grenzen (in German); Proc. 12. Jahrestagung Künstliche Intelligenz; 1989

[Ogden87] W.C. Ogden, A. Sorknes; What Do Users Say to their Natural Language Interface ?; Proc. Conf. Human-Computer Interaction; 1987

[Pareschi87] R. Pareschi, M. Steedman; A Lazy Way to Chart-Parse with Categorial Grammars; Proc. 25th Annual Meeting of the Association for Computational Linguistics; 1987

[Paris91] C.L. Paris; Generation and Explanation: Building an Explanation Facility for the Explainable Expert Systems Framework; in: C.L. Paris, W.R. Swartout, W.C. Mann (eds.); Natural Language Generation in Artificial Intelligence and Computational Linguistics; Kluwer, Boston; 1991

[Pereira80] F.C.N. Pereira, D.H.D. Warren; Definite Clause Grammar for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks; Artificial Intelligence; Vol. 13, No. 3; 1980

[Pereira81] F.C.N. Pereira; Extraposition Grammars; AJCL, Vol. 7, No. 4; 1981

[Pernul93a] G. Pernul, W. Winiwarter, A M. Tjoa; The Deductive Filter Approach to MLS Prototyping; Proc. 9th Annual Computer Security Applications Conf.; 1993

[Pernul93b] G. Pernul, W. Winiwarter, A M. Tjoa; The Entity-Relationship Model for Multilevel Security; Proc. 12th Int. Conf. Entity-Relationship Approach; 1993

[Perrault80] C.R. Perrault, J.F. Allen; A Plan-Based Analysis of Indirect Speech Acts; AJCL; Vol. 6; 1980

[Phillips93] J.D. Phillips; Choosing the Right Word - Lexical Knowledge & Context in Machine Translation; Proc. 1st Conf. of the Pacific Association for Computional Linguistics; 1993

[Przymusinski87] T. Przymusinski; On the Semantics of Stratified Deductive Databases and Logic Programs; in: J. Minker (ed.); Foundations of Deductive Databases and Logic Programming; Kaufmann, Los Altos; 1987

[Quillian68] M.R. Quillian; Semantic Memory; in: M. Minsky (ed.); Semantic Information Processing; MIT Press, Cambridge, MA; 1968

[Ramakrishnan88] R. Ramakrishnan, C. Beeri, R. Krishnamurthy; Optimizing Existential Datalog Queries; Proc. 7th ACM SIGMOD-SIGACT Symp. Principles Database Systems; 1988

[Ramsay87] A. Ramsay; Knowing that and Knowing what; J. Hallam, C. Mellish (eds.); Advances in Artificial Intelligence; Wiley, Chichester; 1987

[Raskutti93a] B. Raskutti, C. Rowles; Using Prosody for Lexical Access to Large Vocabularies; Proc. Workshop Natural Language Processing, 6th Australian Joint Conf. Artificial Intelligence; 1993

[Raskutti93b] B. Raskutti, I. Zukerman; Generating Queries during Cooperative Consultations; Proc. 6th Australian Joint Conf. Artificial Intelligence; 1993

[Rayner93] M. Rayner; Abductive Equivalent Translation and its Application to Natural Language Database Interfacing; Diss., Royal Inst. of Technology, Stockholm; 1993

[Reiter80] R. Reiter; A Logic for Default Reasoning; Artificial Intelligence; Vol. 13; 1980

[Reiter93] E. Reiter, C. Mellish; Optimizing the Costs and Benefits of Natural Language Generation; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Reyle88] U. Reyle; Compositional Semantics for LFG; in: U. Reyle, C. Rohrer (eds.); Natural Language Parsing and Linguistic Theories; Reidel, Dordrecht; 1988

[Rich87] E. Rich; Natural Language Interfaces; in: R.W. Baecker, W.A.S. Buxton (eds.); Readings in Human-Computer Interaction: A Multidisciplinary Approach; Kaufmann, Los Altos; 1987

[Riesbeck78] C. Riesbeck, R.C. Schank; Comprehension by Computer; in: W. Levelt, G.B. Flores d'Arcais (eds.); Studies in the Perception of Language; Wiley, Chichester; 1978

[Ritchie80] G.D. Ritchie; Computational Grammar; Barnes and Noble, New York; 1980

[Robinson82] J.J. Robinson; DIAGRAM: A Grammar for Dialogues; CACM; Vol. 25, No. 1; 1982

[Rowles93] C. Rowles et.al.; The Use of Context in the Understanding of Spoken English; Proc. 6th Australian Joint Conf. Artificial Intelligence; 1993

[Sacca87a] D. Sacca, C. Zaniolo; Magic Counting Methods; Proc. ACM-SIGMOD Conf.; 1987

[Sacca87b] D. Sacca et.al.; The Advanced Database Environment of the KIWI System; IEEE Data Eng., Special Issue on Databases and Logic; Vol. 10, No. 4; 1987

[Sacca87c] D. Sacca, C. Zaniolo; Implementation of Recursive Queries for a Data Language Based on Pure Horn Logic; Proc. 4th Int. Conf. Logic Programming; 1987

[Sacca88a] D. Sacca, C. Zaniolo; Differential Fixpoint Methods and Stratification of Logic Programs; Proc. 3rd Int. Conf. Data Knowledge Bases; 1988

[Sacca88b] D. Sacca, C. Zaniolo; The Generalized Counting Method for Recursive Logic Queries; J. Theoretical Computing Science; Vol. 61; 1988

[Sacerdoti87] E.D. Sacerdoti; Language Access to Distributed Data with Error Recovery; Proc. 5th Int. Joint Conf. Artificial Intelligence; 1977

[Salton83] G. Salton, M.J. McGill; Introduction to Modern Information Retrieval; McGraw-Hill, New York; 1983

[Salton88] G. Salton; A Simple Blueprint for Automatic Boolean Query Processing; Information Processing & Management; Vol. 24, No. 3; 1988

[Sarkar93] A. Sarkar; Extending Kimmo's Two-Level Model of Morphology; Proc. 31st Annual Meeting of the Association for Computational Linguistics; 1993

[Schank74] R.C. Schank, C.J. Rieger; Inference and the Computer Understanding of Natural Language; Artificial Intelligence; Vol. 5; 1974

[Schank75] R.C. Schank (ed.); Conceptual Information Processing; North-Holland, Amsterdam; 1975

[Schank77] R.C. Schank, R. Abelson; Scripts, Plans, Goals and Understanding; Lawrence Erlbaum, Hillsdale; 1977

[Scheer84] A.-W. Scheer; EDV-orientierte BWL (in German); Springer, Berlin; 1984

[Scheer88] A.-W. Scheer; CIM. Der computergesteuerte Industriebetrieb (in German); Springer, Berlin; 1988

[Schröder88] M. Schröder; Evaluating User Utterances in Natural Language Interfaces to Databases; Computers and AI; Vol. 7, No. 4; 1988

[Schubert82] L.K. Schubert, F.J. Pelletier; From English to Logic: Context-Free Computation of Conventional Logical Translation; AJCL; Vol. 8, No. 1; 1982

[Schwanke91] M. Schwanke; Maschinelle Übersetzung (in German); Springer, Berlin; 1991

[Schwarz90] C. Schwarz; Content Based Text Handling; Information Processing & Management; Vol. 26, No. 2; 1990

[Schwartz82] S.P. Schwartz; Problems with Domain-Independent Natural Language Database Access Systems; Proc. 20th Annual Meeting of the Association for Computational Linguistics; 1982

[Schweighofer93a] E. Schweighofer, W. Winiwarter; Refining the Selectivity of Thesauri by means of Statistical Analysis; Proc. 4th Int. Congress Terminology and Knowledge Engineering; 1993

[Schweighofer93b] E. Schweighofer, W. Winiwarter; Legal Expert System KONTERM - Automatic Representation of Document Structure and Contents; Proc. 4th Int. Conf. Database and Expert Systems Applications; 1993

[Schweighofer94] E. Schweighofer, W. Winiwarter; Intelligent Information Retrieval: KONTERM - Automatic Representation of Context Related Terms within a Knowledge Base for a Legal Expert System; Proc. 25th Anniversary Conf. of the Istituto per la documentazione guiridica of the CNR: Towards a Global Expert System in Law; 1994

[Schwind85] B.C. Schwind; Semantikkonzepte in der Künstlichen Intelligenz (in German); in: C. Habel (ed.); Künstliche Intelligenz; Springer, Berlin; 1985

[Seiffert88] R. Seiffert; STUF: Ein flexibler Graphunifikationsformalismus und seine Anwendung in LILOG (in German); in: W. Brauer, C. Freksa (eds.); Wissensbasierte Systeme; Springer, Berlin; 1988

[Selfridge86] M. Selfridge; Integrated Processing Produces Robust Understanding; Computational Linguistics; Vol. 12, No. 2; 1986

[Shieber84] S.M. Shieber; The Design of a Computer Language for Linguistic Information; Proc. Int. Conf. Computational Linguistics; 1984

[Shieber86] S.M. Shieber; An Introduction to Unification-Based Approaches to Grammar; CSLI Lecture Notes, No. 4; Chicago Univ. Press, Chicago; 1986

[Shieber94] S.M. Shieber, Y. Schabes, F.C.N. Pereira; Principles and Implementation of Deductive Parsing; Tech. Rep., TR-11-94, Harvard Univ.; 1994

[Shmueli87] O. Shmueli, S. Naqvi; Set Grouping and Layering in Horn Clause Programs; Proc. 4th Int. Conf. Logic Programming; 1987

[Shmueli88] O. Shmueli, S. Tsur, C. Zaniolo; Rewriting of Rules Containing Set Terms in a Logic Data Language (LDL); Proc. 7th ACM SIGMOD-SIGACT Symp. Principles Database Systems; 1988

[Siekmann90] J. Siekmann; Unification Theory; J. Symbolic Computation; 1990

[Smadja93] F. Smadja; Retrieving Collocations from Text: Xtract; Computation Linguistics; Vol. 19, No. 1; 1993

[Smeaton86] A.F. Smeaton; Incorporating Syntactic Information into a Document Retrieval Strategy: An Investigation; Proc. ACM Conf. Research and Development in Information Retrieval; 1986

[Smedt84] K. de Smedt; Using Object-Oriented Knowledge-Representation Techniques in Morphology and Syntax Programming; Proc. European Conf. Artificial Intelligence; Elsevier, Amsterdam; 1984

[Smith92] K. Smith, M. Winslett; Entity Modeling in the MLS Relational Model; 18th Conf. Very Large Database Systems; 1992

[Sowa84] J. Sowa; Conceptual Structures; Addison-Wesley, Reading, MA; 1984

[Sparck-Jones91] K. Sparck-Jones; Tailoring Output to the User: What Does User Modelling in Generation Mean ?; in: C.L. Paris, W.R. Swartout, W.C. Mann (eds.); Natural Language Generation in Artificial Intelligence and Computational Linguistics; Kluwer, Boston; 1991

[Srihari93] R.K. Srihari, C.M. Baltus; Incorporating Syntactic Constraints in Recognizing Handwritten Sentences; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Steedman85] M. Steedman; Dependency and Coordination in the Grammar of Dutch and English; Language; Vol. 61; 1985

[Steedman87] M. Steedman; Combinatory Grammars and Parasitic Gaps; Natural Language and Linguistic Theory; Vol. 5; 1987

[Steedman91] M. Steedman; Parsing Spoken Language; in: M. Tomita (ed.); Current Issues in Parsing Technology; Kluwer, Boston; 1991

[Steel84] S. Steel; Simplifying Recursive Belief for Language Understanding; Proc. European Conf. Artificial Intelligence; 1984

[Su93] S.Y.M. Su et.al.; OSAM*.KBMS: An Object-Oriented Knowledge Base Management System for Supporting Advanced Applications; ACM SIGMOD; 1993

[Sumita93] E. Sumita et.al.; Example-Based Machine Translation on Massively Parallel Processors; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Sundheim92] B.M. Sundheim; Overview of the Fourth Message Understanding Evaluation and Conference; Proc. Fourth Message Understanding Conf.; 1992

[Suthers93] D. Suthers; Preferences for Model Selection in Explanation; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

[Tauzovich89] B. Tauzovich; An Expert System for Conceptual Modelling; Proc. 8th Int. Conf. Entity-Relationship Approach; 1989

[Tennant83] H.R. Tennant et.al.; Menu-Based Natural Language Understanding; Proc. 21st Annual Meeting of the Association for Computational Linguistics; 1983

[Teorey86] T.J. Teorey, D. Yang, J.P. Fry; A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model; ACM Computer Science; Vol. 18, No. 2; 1986

[Thompson75] F. Thompson, B. Thompson; Practical Natural Language Processing: The REL System as Prototype; in: M. Rubinoff, B. Yovits (eds.); Advances in Computers, Vol. 13; Academic Press, New York; 1975

[Thompson83] C.W. Thompson et.al.; Building Usable Menu-Based Natural Language Interfaces to Databases; Proc. 9th Int. Conf. Very Large Databases; 1983

[Thurmair82] G. Thurmair; Morphologische Verfahren in Information-Retrieval-Systemen (in German); in: H.G. Fischer (ed.); Information Retrieval und natürliche Sprache; Saur, München; 1982

[Tjoa93] A M. Tjoa, L. Berger; Transformation of Requirement Specifications Expressed in Natural Language into an EER Model; Proc. 12th Int. Conf. Entity-Relationship Approach; 1993

[Tomita91] M. Tomita (ed.); Current Issues in Parsing Technology; Kluwer, Boston; 1991

[Trost90] H. Trost, E. Buchberger; Datenbank-DIALOG: How to Communicate with your Database in German (and Enjoy it); Interacting with Computers; Vol. 2, No. 3; 1990

[Tsur86] S. Tsur, C. Zaniolo; LDL: A Logic-Based Query Language; Proc. 12th Int. Conf. Very Large Data Bases; 1986

[Tsur90a] S. Tsur; Applications of Deductive Database Systems; Proc. COMPCON; 1990

[Tsur90b] S. Tsur; Data Dredging; IEEE Data Engineering; Vol. 13, No. 4; 1990

[Tsur90c] S. Tsur, F. Olken, D. Naor; Deductive Databases for Genomic Mapping; Proc. NACLP Workshop Applications of Deductive Databases; 1990

[Ullman89] J.D. Ullman; Principles of Database and Knowledge-Base Systems, Vol. 2: The New Technologies; Computer Science Press, Rockville; 1989

[Ullman90] J.D. Ullman, C. Zaniolo; Deductive Databases: Achievements and Future Directions; ACM SIGMOD Record; Vol. 19, No. 4; 1990

[Uszkoreit86a] H. Uszkoreit; Categorial Unification Grammars; Proc. Int. Conf. Computational Linguistics; 1986

[Uszkoreit86b] H. Uszkoreit; Syntaktische und semantische Generalisierungen im strukturierten Lexikon; Proc. German Workshop Artificial Intelligence and 2. Österreichische Artificial-Intelligence-Tagung; 1986

[Vieille86] L. Vieille; Recursive Axioms in Deductive Databases: The Query-Subquery Approach; Proc. Int. Conf. Expert Database Systems; 1986

[Vieweg94] S. Vieweg et.al.; Enhancing CIM Environments by Security Control; Proc. Int. Conf. Data and Knowledge Systems for Manufacturing and Engineering; 1994

[Vijay-Shanker90] K. Vijay-Shanker, D.J. Weir; Polynomial Parsing of Combinatory Categorial Grammars; Proc. 28th Annual Meeting of the Association for Computational Linguistics; 1990

[Vijay-Shanker94] K. Vijay-Shanker, D.J. Weir; Parsing Some Constrained Grammar Formalisms; Computational Linguistics; Vol. 19, No. 4; 1994

[Wagner85] E. Wagner; Post-Editing SYSTRAN - A Challenge for Commission Translators; Terminologie et Traduction; Vol. 3; 1985

[Wallace84] M. Wallace; Communicating with Databases in Natural Language; Horwood, Chichester; 1984

[Waltz77] D.L. Waltz, B.A. Goodman; Writing a Natural Language Data Base System; Proc. Int. Joint Conf. Artificial Intelligence; 1977

[Waltz78] D.L. Waltz; An English Language Question Answering System for a Large Relational Database; CACM; Vol. 21, No. 7; 1978

[Warren82] H.D. Warren, F.C.N. Pereira; An Efficient Easily Adaptable System for Interpreting Natural Language Queries; AJCL; Vol. 8, No. 3-4; 1982

[Webber83] B.L. Webber; So what Can we Talk about now; in: M. Brady, B. Berwick (eds.); Computational Models of Discourse; MIT Press, Cambridge, MA.; 1983

[Weir88] D. Weir, A. Joshi; Combinatory Categorial Grammars: Generative Power and Relationship to Linear Context-Free Rewriting System; Proc. 26th Annual Meeting of the Association for Computational Linguistics; 1988

[Weischedel83] R.M. Weischedel, N.K. Sondheimer; Meta-Rules as a Basis for Processing Ill-Formed Output; Computational Linguistics; Vol. 9, No. 3-4; 1983

[Weischedel93] R. Weischedel et.al.; Coping with Ambiguity and Unknown Words through Probabilistic Models; Computational Linguistics; Vol. 19, No. 1; 1993

[Wesche88] B. Wesche; Non-Constituent Coordination ohne Funktionale Komposition und Typenanhebung (in German); Proc. 4. Österreichische Artificial-Intelligence Tagung; 1988

[Wheeler86] P. Wheeler; The LOGOS Translation System; Proc. 1st Int. Conf. State of the Art in Machine Translation in America, Asia and Europe; 1986

[White90] G.M. White; Natural Language Understanding and Speech Recognition; CACM; Vol. 33, No. 8; 1990

[Whitelock88] P.J. Whitelock; A Feature-Based Categorial Morpho-Syntax for Japanese; in: U. Reyle, C. Rohrer (eds.); Natural Language Parsing and Linguistic Theories; Reidel, Dordrecht; 1988

[Wilensky83] R. Wilensky; Planning and Understanding; Addison-Wesley, Reading; 1983

[Wilks75] Y. Wilks; An Intelligent Analyzer and Understander of English; CACM; Vol. 18, No. 5; 1975

[Winiwarter93a] W. Winiwarter, A M. Tjoa; Natural Language Interfaces as Integrated Constituents of Deductive Databases; Proc. Symp. Next Generation Database Systems and Their Applications; 1993

[Winiwarter93b] W. Winiwarter, A M. Tjoa; Morphological Analysis in Integrated Natural Language Interfaces to Deductive Databases; Proc. 4th Int. Workshop Natural Language Understanding and Logic Programming; 1993

[Winiwarter93c] W. Winiwarter; Syntactic Analysis for Natural Language Interfaces - the Integrated Deductive Approach; Proc. Workshop Natural Language Processing, 6th Australian Joint Conf. Artificial Intelligence; 1993

[Winiwarter94] W. Winiwarter; Extended CUG for Free Word Order Languages and its Efficient Implementation within an IDA Architecture; Proc. Joint Conf. of 8th Asian Conf. Language, Information and Computation and 2nd Pacific Asia Conf. Formal and Computational Linguistics; 1994

[Winograd72] T. Winograd; Understanding Natural Language; Academic Press, New York; 1972

[Winograd83] T. Winograd; Language as a Cognitive Process, Vol. 1: Syntax; Addison-Wesley, Reading; 1983

[Wittenburg86] K. Wittenburg; Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification Based Formalism; Diss., Univ. Texas, Austin; 1986

[Wittenburg87] K. Wittenburg; Predictive Combinators: A Method for Efficient Parsing of Combinatory Categorial Grammars; Proc. 25th Annual Meeting of the Association for Computational Linguistics; 1987

[Wittenburg91] K. Wittenburg, R.E. Wall; Parsing with Categorial Grammar in Predictive Normal Form; in: M. Tomita (ed.); Current Issues in Parsing Technology; Kluwer, Boston; 1991

[Wong87] S.K.M. Wong et.al.; On Modeling of Information Retrieval Concepts in Vector Spaces; ACM Trans. on Database Systems; Vol. 12, No. 2; 1987

[Woods70] W.A. Woods; Transition Network Grammars for Natural Language Analysis; CACM; Vol. 13; 1970

[Woods72] W.A. Woods, R.M. Kaplan, B. Nash-Webber; The Lunar Sciences Natural Language Information System; Bolt Beranek and Newman, Cambridge, MA.; 1972

[Woods73] W.A. Woods; An Experimental Parsing System for Transition Network Grammars; in: R. Rustin (ed.); Natural Language Processing; Algorithmics Press, New York; 1973

[Woods75] W.A. Woods; What's in a Link: Foundations for Semantic Networks; in: D.G. Bobrow, A. Collins (eds.); Representation and Understanding: Studies in Cognitive Science; Academic Press, New York; 1975

[Woods78] W.A. Woods; Semantics and Quantification in Natural Language Question Answering; in: M. Yovitz (ed.); Advances in Computers; Vol. 17; Academic Press, New York; 1978

[Woods80] W.A. Woods; Cascaded ATN-Grammars; AJCL; Vol. 6, No. 1; 1980

[Wu93] D. Wu; An Image-Schematic System of Thematic Roles; Proc. 1st Conf. of the Pacific Association for Computational Linguistics; 1993

[Young85] M.A. Young, P.J. Hayes; Automatic Classification and Summarization of Banking Telexes; in: C.R. Weisbin (ed.); Artificial Intelligence Applications; IEEE Press, Washington; 1985

[Yuan93] L.Y. Yuan, J.-H. You; The Alternating Semantics for Default Theories and Logic Programs; Proc. 6th Australian Joint Conf. Artificial Intelligence; 1993

[Zaniolo85] C. Zaniolo; The Representation and Deductive Retrieval of Complex Objects; in Proc. 11th Int. Conf. Very Large Data Bases; 1985

[Zaniolo88] C. Zaniolo; Design and Implementation of a Logic Based Language for Data Intensive Applications; Proc. Int. Conf. Logic Programming; 1988

[Zaniolo90a] C. Zaniolo; Architectures of Deductive Database Systems; Proc. COMPCON; 1990

[Zaniolo90b] C. Zaniolo; Deductive Databases - Theory Meets Practice; Proc. 2nd Int. Conf. Extending Database Technology; 1990

[Zaniolo90c] C. Zaniolo; Rule Rewriting Methods in the Implementation of the Logic Data Language LDL; R.A. Meersman, Z. Shi, C.-H. Kung (eds.); Artificial Intelligence in Databases and Information Systems; North-Holland, Amsterdam; 1990

[Zeevat88] H. Zeevat; Combining Categorial Grammar and Unification; in: U. Reyle, C. Rohrer (eds.); Natural Language Parsing and Linguistic Theories; Reidel, Dordrecht; 1988

[Zsolnai87] S. Zsolnai, H. Trost; Towards Automatic Semantic Classification for a Natural Language Understanding System; Proc. 3. Österreichische Artificial-Intelligence-Tagung; 1987

[Zukerman93] I. Zukerman, R. McConachy; Generating Concise Discourse that Addresses a User's Inferences; Proc. 13th Int. Joint Conf. Artificial Intelligence; 1993

List of Figures

List of Tables