Lecture Notes in Artificial Intelligence
1735

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Jan Willers Amtrup

Incremental Speech Translation

ISpring»

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA Jo¨rg Siekmann, University of Saarland, Saarbru¨cken, Germany

Author

Jan Willers Amtrup
New Mexico State University, Computing Research Laboratory
P.O. Box 30001, Las Cruces, NM 88003, USA
E-mail: jamtrup@crl.nmsu.edu

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Amtrup, Jan:
Incremental speech translation / Jan Willers Amtrup. - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; 1735 : Lecture notes in artificial intelligence)
ISBN 3-540-66753-9

CR Subject Classification (1998): I.2.7, I.2, F.4.2-3

ISBN 3-540-66753-9 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN: 10750071 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Foreword

Human language capabilities are based on mental procedures that are closely linked to the time domain. Listening, understanding, and reacting, on the one hand, as well as planning, formulating, and speaking, on the other, are performed in a highly overlapping manner, thus allowing inter-human communication to proceed in a smooth and fluent way.

Although it happens to be the natural mode of human language interaction, incremental processing is still far from becoming a common feature of today's language technology. Instead, it will certainly remain one of the big challenges for research activities in the years to come. Usually considered difficult to a degree that renders it almost intractable for practical purposes, incremental language processing has recently been attracting a steadily growing interest in the spoken language processing community. Its notorious difficulty can be attributed mainly to two reasons:

- Due to the inaccessibility of the right context, global optimization criteria are no longer available. This loss must be compensated for by communicating larger search spaces between system components or by introducing appropriate repair mechanisms. In any case, the complexity of the task can easily grow by an order of magnitude or even more.
- Incrementality is an almost useless feature as long as it remains a local property of individual system components. The advantages of incremental processing can be effective only if all the components of a producer-consumer chain consistently adhere to the same pattern of temporal behavior. Particularly for inherently complex tasks like spoken language translation the issue of incremental processing cannot be treated as a local phenomenon. Instead it turns out to be intrinsically tied to fundamental questions of the overall system architecture, thus requiring a global perspective and the ability to create and maintain a sufficiently ambitious experimental environment.

If, despite these difficulties, a first prototypical solution for the incremental translation of spoken dialogues is presented here, two fundamental ideas have contributed most to this remarkable success: the use of a chart as a uniform data structure throughout the system and the rigorous application of results from graph theory that eventually allowed the complexity of the task to be reduced to a manageable degree.

This combination of contributions enables us for the first time to observe how a machine translation of natural language utterances evolves over time as more and more input becomes available. It certainly is much too early to risk a direct comparison with human interpretation capabilities, but certainly this book puts forward a benchmark against which other solutions will have to be measured in the future.

October 1999 Wolfgang Menzel

Preface

Automatic speech recognition and processing has received a lot of attention during the last decade. Prototypes for speech-to-speech translation are currently being developed that show first impressive results for this highly complex endeavor. They demonstrate that machines can actually be helpful in communicating information between persons speaking different languages. Simple tasks, e.g. the scheduling of business appointments or the reservation of hotel rooms and air travel tickets, are within reach.

Needless to say, the power of these prototypes is far from being equal to the human abilities for speaking, hearing, understanding, and translating. Performing the translation of speeches or free dialog at a high level is one of the most ambitious goals of scientists in the natural language processing domain. Several major areas of research have to be fruitfully combined to create even basic systems and demonstrators. Progress is needed regarding each of the several steps that are performed while creating a translation of an utterance spoken by a speaker, involving fields like acoustics, speech recognition and synthesis, prosody, syntactic processing, semantic representation, contrastive studies for translation, and many others.

This book starts from an outside view to speech translation, a view that does not concentrate immediately on one of the tasks we mentioned. The main motivation for the research presented in this monograph is the fact that humans understand and translate while they are still hearing. This *incremental* operation is in part responsible for the relative ease with which we handle certain tasks, like simultaneous interpreting or simply following a conversation at a party with a high level of background noise.

The application of this paradigm to automatic speech processing systems seems to be a natural thing to do, yet it has serious consequences for the implementation of individual components and the system as a whole. The simple demand "Start analyzing while the input is still incomplete" in some cases requires difficult modifications to the algorithms employed for certain tasks.

We think that incremental, modular systems require careful attention as to how they are composed from individual components. Interfaces and their use become more crucial if a component is to deliver not only a small set of final results (in many cases exactly one result), but a continuous stream of hypotheses. Thus, the realization of incrementality also initiated the use of an integrated data structure (the layered chart) and the use of a uniform formalism for all modules.

The first chapter introduces incrementality and provides a motivation for its use in automatic speech translation. Chapters 2 and 3 give the necessary theoretical foundation to describe the system presented here adequately. In particular, chapter 2 focuses on graph theory and its application to natural language processing. We believe that a wide range of phenomena and algorithms for NLP can be most adequately described (and most easily understood) in terms of a small subset of graph theory. Chapter 3 presents the uniform formalism that is used throughout the system: Typed Feature Structures.

Chapter 4 and 5 describe the system that provides the background for this book. In our opinion, interesting architectural paradigms cannot be shown in isolation from an actual system implementing these paradigms. The system MILC demonstrates the feasibility of employing incrementality to a complete speech translation system. We describe how the three architectonic principles incrementality, integrity, and uniformity are used to compose a non-trivial system, and demonstrate its performance using actual speech data. Finally, chapter 6 provides a conclusion.

The research described in this book was performed while the author was a research scientist in the Natural Language Systems Group within the Computer Science Department of the University of Hamburg, Germany. The German version of this monograph was accepted as dissertation by its CS department.

I am indebted to the teachers that provided a major part of my education in computer science. Walther von Hahn introduced me to Natural Language Processing. Gu¨nther Go¨rz attracted me to syntactic parsing and the architecture aspects of NLP systems. Finally, Wolfgang Menzel discussed large parts of my work and contributed a lot as my primary thesis advisor.

The Natural Language Systems group in Hamburg provided an excellent research environment, from which I benefited during the five years that I worked there. I wish to thank all colleagues for their cooperation, especially Andreas Hauenstein, Henrik Heine, Susanne Jekat, Uwe Jost, Martin Schro¨der, and Volker Weber.

While in Hamburg, I worked as a Verbmobil project member for more than three years. This enabled me to have many fruitful discussions and to cooperate with several people, especially Jan Alexandersson, Thomas Bub, Guido Drexel, Walter Kasper, Marcus Kesseler, Hans-Ulrich Krieger, Joachim Quantz, Birte Schmitz, Joachim Schwinn, Jo¨rg Spilker, and Hans Weber.

Some of the work presented here (including most of the experiments) was finished at the Computing Research Laboratory at New Mexico State University. Special thanks to Karine Megerdoomian, Sergei Nirenburg, and Re´mi Zajac.

August 1999 Jan Willers Amtrup

List of Tables

List of Algorithms

List of Figures

Contents

Chapter 1

Introduction

In this chapter, we will first provide a formal definition of incrementality. The consequences that arise when using incremental mechanisms in natural language processing systems are discussed. We will discuss the ability of humans to process speech incrementally as a major motivation for our work. Ultimately, we present a coarse sketch of the architecture of the system we develop here.

1.1 Incremental Natural Language Processing

Three properties of natural language processing systems have gained increasing importance over the last decade: Modularity, parallelism and incrementality. In the NLP field as well as in any other, the application of techniques from software engineering, like modularity, allows for the construction of large systems (Sommerville, 1996). Modularity promotes reusability, stability, and in the long run also the efficiency of individual components. Furthermore, only modular research systems are suitable for scaling up in the direction of commercially relevant applications (Zajac, Casper and Sharples, 1997). The distribution of algorithms over several processors by means of employing parallel procedures is, in our view, essential for constructing large systems which show a sufficiently high performance. The incorporation of parallel operation into modules for natural language processing has been a side issue so far, although there are very interesting approaches, especially in the field of massively parallel systems (Kitano, 1994).

The application of incremental methods on a large scale is still in its infancy, despite considerable efforts. This is particularly true as far as the consequences for the architecture of NLP systems are concerned.

In many cases, the definition of terms and their differentiation are a source of misunderstanding when it comes to comparisons between systems. Especially if different architectural facets are related to each other, it seems eminently important to have a common set of concepts. In the following, we will give some basic definitions to unambiguously describe the subject of this research. Naturally, the

J.W. Amtrup: Incremental Speech Translation, LNAI 1735, pp. 3-24, 1999.

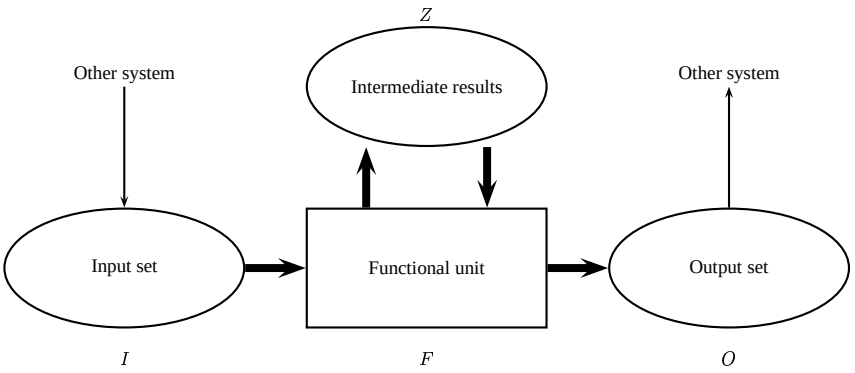


Figure 1.1. The architecture of a functional unit within a modular system

definition of incrementality is the most important. We will present this concept in a formal way.

First, one has to distinguish between a *monolithical* and a *modular* system. Monolithical systems have no separable subparts according to functional or other criteria. They seem to have no inner structure on a higher level than that of atomic instructions. On the other hand, modular systems contain several functional units, which are isolated from each other for the most part. Information is exchanged using well-defined interfaces. Figure 1.1 shows the abstract schema of such a functional unit, which we will use later to define incrementality formally. The composition of such modules in a certain way comprises a complete system, which is modular in nature. All systems that we use and refer to in this work are modular.

Modularity is a structural criterion for the classification of systems. The question of how many input elements can be processed at a given point in time is a relevant aspect for the system behavior. A *sequential* system is able to process only one piece of information at any point in time. On the other hand, a *parallel* system may handle more than one input item. Consequently, there is at least one situation in which two or more chunks of data are being processed. This simultaneous operation may surface in an *intra-modular parallelism*. Using this paradigm, the functional unit representing one module is divided into more than one processing block. These blocks are able to work at least partially autonomously. In principle, it must be distinguished between parallelism produced by algorithmic means and data-driven parallelism. One example for algorithmic parallelism is the parallel unification of feature structures (Hager and Moser, 1989). Data-driven parallelism is created by

providing multiple processing units that operate on different subsets of input data and intermediate results. Examples of this are the division of an input utterance into smaller parts or the partitioning of a grammar (Amtrup, 1992). *Memory-based* or *example-based* algorithms, which compare input elements to instances that have previously been presented to them, could also fall into this class.

The counterpart to this kind of "internal" parallelism is given as *inter-modular* parallelism. This type of parallel operation is given if two different functional units or modules operate at the same time (e.g., if the syntactic and semantic analysis are carried out in separate modules). To ensure this, there have to be input elements for both modules at the same time. In general, this is only possible if incremental processing is obeyed.

Simply stated, *incrementality* means that input data is not considered in its entirety, but in a piecemeal fashion. This means that processing starts with individual parts of the input, before the input is completely known to the functional unit. The kind of incrementality that is most important for our purposes is called *chronological incrementality* (or Left-Right (LR-) incrementality). LR-incremental systems operate on data which extend in time, as given in natural language processing situations by the sequence of input words. For systems that analyze spoken language, the chronological nature is given through the dependency of the input signal on time. In principle, other forms of incrementality are plausible (e.g. according to structural criteria), however, these types will play no further role in this investigation.[1]

The maxim of incremental language processing for analysis is to divide the input data into segments of uniform or heterogeneous length. On the level of speech signals, this segmentation is usually provided by the scan rate of preprocessing, in most cases 10ms. On higher levels, there exists a non-uniform order in time which is defined by the endpoints of word hypotheses or phrasal hypotheses.[2] A system becomes LR-incremental by observing the strict order in which input elements are processed. Backtracking in time is either not allowed at all, or only for small time frames.

For reasons of relevancy, we will now define incrementality and its subconcepts of incremental input, incremental output and incrementality of a system in a formal way. We take the architecture depicted in Figure 1.1 as a model for an incremental device (cf. Amtrup, 1997c). Let F be a functional unit that produces some result by applying operations to a given input. The input data to F is given as a set I of input elements $A_i, i \in \{1, \ldots, n\}$. The A_i each arrive in I according to some order of arrival, i.e. there is a stream of input elements into I. All A_i are independent of each other in that their order of arrival is not predetermined by their content.[3]

[1] Incrementality is used in compiler construction to compile only those parts of programs which actually changed during a modification (Marchetti-Spaccamela, Nanni and Rohnert, 1992).

[2] In many cases, this implies also a structural incrementality given by the compositionality of several language processing methods.

[3] There may exist an external dependency which is caused by the way in which previous functional units work. However, these external reasons are irrelevant in this argument.

The set O is made up of final results of F and is constructed analogously to I. O consists of output elements $B_j, j \in \{1, \ldots, m\}$. Intermediate results $C_k, k \in \{1, \ldots, p\}$ are stored in a set Z. We assume that no processing is done in I or O, so that each operation in F either changes the content of Z or produces a new output element in O. A new intermediate result in Z is calculated by applying a function f, while a different function g handles the creation of output elements in O:

$$C_k := \quad f(D_k), \quad D_k \in I^* \times Z^* \tag{1.1}$$

$$B_j := \quad g(D_j), \quad D_j \in I^* \times Z^* \tag{1.2}$$

Typically, g is a simple selection function that licenses intermediate results from Z as output and distributes them to the set of output elements O.

The incremental arrival of input items is modeled with the assumption that the elements A_i in Figure 1.1 are delivered in an arbitrary order and show up in I, i.e. there is a stream of input elements filling up I. Each result B_j, which is produced by F, is handed to the system attached to the output channel of F, immediately after it has entered O. A LR-incremental mode of operation is ensured by the additional property of the input stream to supply the hypotheses in chronological order.[4]

Now, we can define the state of the system F by examining the contents of the three sets I, O and Z:

$$\begin{aligned}&\text{At every time } t \text{ with } 0 \leq t < \infty \text{ the state of processing in } F\\ &\text{is given as the actual contents of } I(t), O(t) \text{ and } Z(t).\end{aligned} \tag{1.3}$$

There are two extreme situations F may be in. In the initial situation all three sets are empty:

$$I(0) = \emptyset \wedge O(0) = \emptyset \wedge Z(0) = \emptyset \tag{1.4}$$

On the other hand, processing is complete if no changes occur on these sets. All input elements have been taken into consideration, all output elements have been created:

$$\begin{aligned}t_{\text{End}} := t: \quad & I(t) = \{A_1, \ldots, A_n\} \wedge \\ & O(t) = \{B_1, \ldots, B_m\} \wedge \\ & Z(t) = \{C_1, \ldots, C_p\}.\end{aligned} \tag{1.5}$$

We can distinguish among three different types of incrementality[5]:

[4] For example, the word hypotheses delivered by a speech recognizer are usually sorted according to their end frames, providing a partial order among them.

[5] Cf. Kilger (1994) for an informal account of these types.

- Incremental input. The system F starts to work on input items without waiting for all input elements being present:

$$\exists t : \#(I(t)) \neq n \wedge Z(t) \neq \emptyset \tag{1.6}$$

  An example application might be a NLP system capable of answering Yes-No questions that starts to process input as soon as it arrives. However, the actual answer may depend on the last input element, thereby preventing the output of a result before all input has been assessed.
- Incremental output. Some elements of the output are presented before all processing is complete, i.e. before I and Z both reach their final content:

$$\exists t : \#(I(t)) \neq n \wedge \#(Z(t)) \neq p \wedge O(t) \neq \emptyset \tag{1.7}$$

  You will get an impression of incremental output by thinking of a system that reads out sentences: It gets a whole sentence and reads it out while computing speech signals step by step.
- Incremental system. An incremental system works on parts of the input and starts to issue output elements before the input is complete:

$$\exists t : \#(I(t)) \neq n \wedge O(t) \neq \emptyset. \tag{1.8}$$

  An example for an incremental system could be a parser which reads the input word for word. Each item is processed as it arrives and partial hypotheses about growing portions of the input are delivered.[6]

It is obvious that incrementality means processing input data in small portions. In the extreme case, the output produced by a system may actually overlap the input at hand. A crucially important measure for the description of an incremental system is the *increment size*, i.e. the fraction of input data which is used as a portion for processing. The size of increments may vary from very small (frames of 10ms length) to relatively large (complete utterances in dialogs). The actual extension in time may differ for individual increments, of course. Even if hypotheses are delivered in strict chronological order, there are cases where different hypotheses extend over different intervals in time, e.g. in the case of constituents of variable length that are delivered by a parser. In order for an incremental system to be successful, it is essential to adjust the increment sizes of the various components in an adequate way.

A simple method to assess the combination of components and increment sizes is to measure the delay. The *delay* between two components is defined as the difference between the current points of processing of two modules, i.e. the difference between the time-stamps of the input items which are processed at that time (Pyka, 1992c). In the ideal case, if a module is able to process each input element immediately and quickly, the delay is determined by the increment size of the preceding component. The overall delay of the system can be calculated as the sum of the

[6] Incremental input and incremental output can be combined without yielding an incremental system. In such systems there is a time t such that $\#(I(t)) = n \wedge \#(Z(t)) \neq p \wedge O(t) = \emptyset$, i.e. input and output do not overlap.

individual delays of the components in this case.[7] This ideal case cannot be found in real systems, the calculation of the system delay can at best be used to get a qualitative estimate (cf. Weber, Amtrup and Spilker, 1997). Usually, the time a component spends for processing an input element (and even transmitting it) is not negligible; instead, these times have a great impact on the behavior of an application. Increment sizes which are chosen to be too small may lead to a processing schema without enough context. This may jeopardize the success of the processing in general, e.g. in the case of a word-to-word translation that is unaware of a broader context. Moreover, components that invest a big amount of processing into each input element can no longer react in adequate time because they become overloaded with input. On the other hand, increments that are too big may lead to suboptimal load balancing for modules in a distributed, parallel system. The choice of increments for the system presented here tries to use sizes as small as possible in order to minimize delay, without affecting quality or performance adversely.

It is desirable to extend the concepts described above, which assumed that the set of input elements is finite, to cover continuous systems, as we may formulate a restricted analogy to the human speech understanding apparatus, which is in permanent operation during the lifespan of a person. However, this extension is only valid in narrow limits. First of all, it is to say that a continuously working system trivially meets the colloquial definition of incrementality, "Produce output, while the input is not yet complete", simply because the input will never be complete.

A correct, useful handling thus requires the identification of interrelated ranges of input and output. Considerations about the properties of a system are then restricted to intervals within the input stream which constitute a range containing logically dependent elements. Within these individual intervals, the aforementioned definitions can be applied. To determine the borders of such ranges is usually a matter of subjective judgment in each individual case. If formal systems are to be investigated, the intervals may be constructed by calculating the symmetric closure of f and g w.r.t. the input data and the set of intermediate results. An interval in time is locally connected if f and g operate inside of the interval and if the interval is minimal. Here the definitions stated earlier are applicable. If biological systems are considered, e.g. in the investigation of the human natural language processor, the formulation of interval borders becomes more difficult. Even obvious demarcations can be misleading. For example, sleep would be ordinarily seen as definite borderline for intervals of operation of the human speech understanding. However, this process is indeed dependent on experiences which extend much further into the past, e.g. it depends on the frequency with which certain words have been heard (Shillcock, 1990).

Thus, the restriction to specific intervals in time always entails an abstraction regarding certain properties of the objects under investigation and the question of granularity becomes more complex. In the case at hand, the upper bound is given

[7] We would like to reemphasize that we are only concerned with incrementality based on the concept of time. Concepts like the delay of a component are useless for systems working incrementally according to structural criteria.

by individual utterances of a cooperative speaker within an appointment scheduling dialog. All knowledge sources that would be able to describe broader contexts, e.g. a dialog model, or a sophisticated world model, etc., are not considered, although they clearly have an influence on the interpretation of a turn of a speaker.

The incremental design of a system exclusively yields disadvantages (Ausiello *et al.*, 1991). Apart from the fact that incremental algorithms are most often more complex than their non-incremental counterparts[8], there is no possibility of a global optimization due to the lack of right context. In the case of the processing relevant for this work — the linguistic understanding of word graphs (cf. Section 2.2) — this surfaces as the fact that the acoustically best-rated path through a word graph is not known. In the extreme case, it may even not be present.[9]

A further consequence of incrementality may be a drastic increase in the size of search spaces of individual components and of the complete system. Again, this can be shown using the production of word graphs in a speech recognizer and their subsequent analysis, e.g. in a syntactic parser. If word graphs are constructed incrementally, the elimination of dead ends becomes impossible. *Dead ends* are word hypotheses that cannot be extended by further hypotheses due to the lack of sufficiently high-rated words. In conventional recognizers, these hypotheses are accounted for in a backward search after recognizing the whole utterance. As a consequence of the presence of dead ends, the subsequent parser is unable to decide whether a given word hypothesis could ever be part of a complete analysis describing a path covering the whole input. Thus, the parser spends time and effort for the processing of partial paths that probably will never lead to a global final state.

Viewed in isolation, incrementality is obviously disadvantageous. However, if considered for the combination of modules within a system, the situation becomes different. Incrementality has the potential of opening possibilities in architectonic and algorithmic decisions, which will probably increase both quality and performance of processing. Experiments we will describe later (see Section 5.3) show that the exploration of techniques of this kind enables the incremental processing of speech to be almost as efficient as conventional methods are nowadays.

A direct result of the incremental nature of modules in a system is the opportunity to introduce inter-modular parallelism, which can immediately be used to increase performance. Only if one functional unit begins to produce output elements while the processing as a whole is not completed, and subsequent functional units begin to work on these elements incrementally, there is the possibility of both modules running in parallel.[10]

Of course, this does not mean that modular, incremental systems always work in parallel. Neither does it mean that non-incremental systems always lack paral-

[8] At least it is usually not easy to formulate the necessary update procedures efficiently (cf. Ramalingam and Reps, 1992).

[9] Unknown means that a prefix of the acoustically best path is present, but other prefixes are better so far. Given the possibility of very long words, there may even be the situation that the prefix has not been constructed to the current point in time. In this case, the prefix is not even present.

lelism. Intra-modular parallelism is always possible. In fact, almost all programs are currently being executed in parallel on a very low level, that of processor-pipelining.

Interactivity is the second dimension of architectonic modifications that is opened through the employment of incremental techniques. In addition to the main data stream between modules, which flows in one direction (*feed forward*), interactive systems may be equipped with streams operating in the opposite direction. The motivation for installing such a type of interaction is always to let one module influence the mode of operation of another module participating in the computation (*top-down interaction*). Naturally, the influence can only take place successfully if the processing of input data in the target module has not yet been finished. The choice of increment sizes is crucial here; even a system working on spoken dialogs that operates on whole utterances at once can be said to be incremental on the turn level, as long as e.g. the dialog management produces predictions about the structure or content of following turns. By establishing feedback mechanisms between modules additional functionality can be gained, for example by rejecting hypotheses or by generating predictions.[11]

If, for example, the architecture principle of interactivity is applied to incremental speech recognition with word graphs, this could lead to an interleaved schema of recognition and syntactic parsing. The effect of this measure would be to reduce the search space of the recognizer. A necessary prerequisite for interaction in this way is the synchronization of recognizer and parser, which have to operate alternatingly on the same set of hypotheses. The parser licenses partial paths in the word graph by ensuring that there is the possibility that they may be further expanded. If for any partial path no syntactically valid continuation remains present, the corresponding branch of the search space within the recognizer may be cut off. This interaction can be used to enhance the performance of a word recognition module (Hauenstein and Weber, 1994; Fink, Kummert and Sagerer, 1994).

In general, the statement holds that only the early construction and verification of hypotheses in components may have the effect of eliminating parts of search spaces in preceding modules. Furthermore, only the interleaved mode of operation utilizing a tight interaction pattern enables the evaluation of predictions. Predictions generate more information than just a binary decision which might be used to prevent further effort for the pursuit of a futile hypothesis. They are capable of setting targets for other modules. This results in a modification of the structure of the affected search space, which in turn could lead to a construction of results in a different order. Related to the example at hand, a possible approach would be to generate the set of syntactically valid continuation words at each point in time, thereby providing the word recognizer with a dynamically created, restricted vocabulary. The immediate consequence of this technique does more than simply prevent

[10] There is one exception from this general rule: A module may produce two identical copies of its output that are distributed to two other modules that perform independent functions on these data. These two modules can work in parallel.

[11] Feedback may also lead to oscillation and resonance — phenomena which might pose serious problems.

certain branches of the search space from being explored. Rather, the shape of the search space itself is changed (Weber, 1995, p. 64).

The system presented here does not use interaction in terms of the generation of predictions. The possible influence that modules may inflict upon each other in the incremental paradigm is demonstrated using the analysis and translation of idiomatic expressions. This task is performed separately using fast incremental graph search algorithms. Identified idioms result in the modification of scores of word hypotheses and the phrasal constituents they support, changing the way in which they are treated in the deep linguistic processing modules.

1.2 Incremental Speech Understanding

We have already mentioned that the research presented in this book is oriented towards the goal of designing and implementing incremental algorithms for machine translation of spoken language in analogy to mechanisms of human speech and language understanding. Everyone can experience the effects incrementality has for hearing and understanding speech by noticing how words are recognized even before they are completely spo...[12]

One theory that attempts to describe and explain this early phase of auditive speech recognition is called the *cohort model*, proposed by Marslen-Wilson (Marslen-Wilson and Welsh, 1978; Marslen-Wilson and Tyler, 1980). The underlying hypothesis is that the recognition of a word is a competitive selection process among a set of several candidates. The main criterion for selection is the incoming speech signal. The first 150ms of signal already suffice to assemble the initial *cohort*, a set of words which are compatible with the input signal at hand. During further hearing members of the cohort are eliminated, as they no longer match the acoustic input well enough. Finally, when only one member of the cohort remains, the theory postulates that word to be the one spoken. The *lexical decision* is performed, the word is recognized. This kind of competitive word recognition leads to situations in which a word can be anticipated before the hearer has perceived the complete input signal. This phenomenon can be measured easily in *cross-modal priming experiments*. In these experiments, stimuli (here: fragments of words) are offered to a subject, and it is examined whether the presentation leads to a better (i.e., in general: faster) recognition of selected target concepts. To prevent mutual influences between stimulus and target concept, the latter is given using a different modality (e.g. on a screen). If a target concept which is semantically close to a stimulus is being recognized faster than a semantically unrelated concept, one can conclude that the meaning of the stimulus was activated, that the word in total was recognized. Zwitserlood (1989) uses fragments like [capt] and shows a better reaction in subjects using the semantically related [ship]. This happens even though the stimulus [captain] was not complete.

[12] Of course, we can only simulate the auditive channel using this medium.

This early version of the cohort model is not free of problems, however. In particular, two areas of the theory can be criticized. The importance of the word initial cohort and the unconditional removal of candidates from the set of potentially recognized words are the reasons for not accounting for certain effects. According to the theory, a very early interval of the speech signal is used to create the initial cohort during the *lexical access phase*. In principle, this implies that a segmentation of the input into individual words has to be carried out before word recognition starts. However, the division of the acoustic signal into distinct words is not a trivial task, neither for artificial systems for word recognition, whose accuracy decreases significantly as the input changes from isolated words with distinct pauses between them to continuous speech, nor for humans, who sometimes misunderstand

It's easy to recognize speech **(1.1)**

for

It's easy to wreck a nice beach[13] **(1.2)**

The assumption that segmentation happens prior to recognition is also not compatible with the recognition of sub-words in stimuli. This phenomenon occurs when subjects hypothesize parts of a word as words in their own right. For example, if the stimulus [trombone] is presented, subjects also recognize [bone], although it is embedded in the original word and thus should not give rise to an initial cohort, if one assumes a preceding segmentation. Moreover, a segmentation isolating [bone] as a word is unlikely to succeed because the first segment [trom] is not a valid word.

The second problem refers to the next stage of the word recognition model. This *lexical selection phase* reduces the size of the initial cohort according to acoustic evidence (and other kinds of knowledge) in order to eventually be able to select the intended word. According to Marslen-Wilson and Welsh (1978), this membership decision is binary; candidates are removed from the cohort if their models in the lexicon deviate from the actual acoustic realization. This mechanism entails that perception errors should, in principle, lead to the failure of the recognition task. Those errors could be caused by suboptimal articulation or the presence of background noise. In contrast to this model, the *phoneme restoration effect* (cf. Warren, 1970) demonstrates that incomplete words are being recognized without problems, words in which parts of the acoustic stimulus were overlapped or even replaced by noise. Even if the disturbed segment is positioned in the early part of a word — prior to the decision point at which the size of the cohort could be reduced to one — the recognition does not fail.

A modification of the cohort theory by Marslen-Wilson (1987) makes for a more suitable description of the performance of humans in the early phases of word recognition, lexical access and lexical selection. The competition for membership in a cohort is now described in terms of scores for word hypotheses which are not restricted to being activated at the onset of a word. This enables a conclusive explanation of why one correctly recognizes [cigarette], even if [shigarette] has been uttered.

[13] cf. Shillcock (1990).

The phoneme restoration effect also gives evidence for the fact that the perception of acoustic signals is strongly supported by lexical knowledge. The means to demonstrate this are *signal detection tests*. Here, stimuli are classified by subjects based on whether or not a phoneme has been overlaid or replaced by noise. Through special considerations of the experimental setting, post-perceptive processes are suppressed. Thus, the influence of the lexicon in signal recognition can be determined. This influence manifests itself in a decreasing quality of the discrimination of replacement and overlay when words are heard as opposed to the hearing of nonwords. Samuel (1990) concludes that either acoustic perception and lexicon are both part of one and the same functional unit (they constitute a module), or that both are modules on their own which interact heavily — they perform top-down interaction in the terminology used here.

Higher levels of linguistic processing (syntax, semantics) in general have no influence on perception. This extends to the lexical access phase of the cohort model. It can generally not be shown that syntactic restrictions play a role when inserting a word into a cohort. The modularity hypothesis of Fodor (1983) seems to be valid here. Consider the presentation of

They all rose **(1.3)**

Besides the verbal interpretation, the nominal reading is also activated, which was shown by cross modal priming experiments (Tanenhaus, Leiman and Seidenberg, 1979, after Shillcock and Bard, 1993). However, this result holds only for open-class words. If the current syntactic context permits us to hypothesize a closed-class item, only the words belonging to the closed class are activated in the cohort (Shillcock and Bard, 1993). Accordingly, the nominal reading of the homophones would and wood is suppressed in contexts like

John said that he didn't want to do the job, but his brother would **(1.4)**

This phenomenon actually seems to be caused by the membership in a closed or open word class. Ying (1996) shows that homophones are articulated differently depending on the context, but Shillcock and Bard (1993) argue that the different realizations could not be distinguished by phoneticians, and that they were articulated identically.[14]

The lexical selection phase depends much more on evidence from other levels than lexical access.[15] During the selection process, a semantic influence that is generated by the preceding context can be viewed as essential for the modification of scores of different word hypotheses within a cohort. This is demonstrated using

[14] A stronger position would be possible if the realizations would have been interchanged in the presentations.

[15] Classical theories like the cohort model or TRACE (McClelland and Elman, 1986) usually assume a division of the recognition knowledge into separate layers (phonological, syntactic, semantic). Gaskell and Marslen-Wilson (1995) propose the incorporation of all possible influences into a single, unified mechanism, a recurrent neural network.

priming experiments. A word that is semantically supported by context is shown to have a higher relative activation than non-supported words. Assume that the context

The men stood around the grave. They mourned at the loss of their . . . **(1.5)**

is presented to subjects. The stimulus [cap] results in a priming of [captain] over [capital], although both words share the same acoustic prefix (Zwitserlood, 1989). Again, there exist differences in the processing of open- and closed-class items (cf. Friederici, 1992).

The results of psycholinguistic research displayed so far exhibit an acceleration of word recognition and a reduction of lexical ambiguity as a result of the incrementality of human speech understanding. The effects of incremental processing also extend into higher levels, however. There seem to be interactions between the processes involved here that enable humans to cope with the extremely high degree of ambiguity by excluding alternatives early, by courageously following analysis paths and by making decisions without a complete knowledge of facts.[16]

To cite just one example: Niv (1993) argues that certain decisions about the syntactic functions of constituents are drawn using information from a non-syntactic domain. A principle which he uses to illustrate this effect is called: *Avoid new subjects*. It covers the reasons for certain preferences of syntactically equivalent embeddings. Consider

The maid disclosed the safe's location { a) to the officer / b) had been changed **(1.6)**

The variant b) not only requires the detection of a reduced relative clause, but also the categorization of the safe's location as a subject, whereas a) uses a direct object reading. Experiments show that subjects prefer reading a), which leads Niv (1993) to the conclusion that a subjective reading is avoided if the nominal phrase in question has not yet been introduced into the discourse.

The importance of incremental processing for human communication behavior becomes obvious when behavioral patterns in dialog situations are investigated. Dialog partners interrupt each other quite often, if, e.g., the violation of a presupposition was recognized. This behavior is only possible through the incremental nature of human language understanding. This becomes extremely clear considering the task of simultaneous interpreting, e.g. at conferences (Künzli and Moser-Mercer, 1995). A conference interpreter cannot afford to wait for a speaker to complete his or her utterance before the production of the translation starts, simply for reasons of time and workload (Anderson, 1994; Klein, Jekat and Amtrup, 1996). Thus,

[16] There is a reason why humans have difficulty understanding *garden path sentences* like "The horse raced past the barn fell.". The investigation of deterministic methods of natural language understanding has thus been a prominent matter for a long time (Marcus, 1980). The understanding of syntactic ambiguities, for example, sometimes poses almost unsolvable problems for the developer of a NLP system.

interpreters not only carry out two tasks simultaneously, but both tasks are additionally strongly correlated. Chernov (1994) reports that interpreters hear and speak simultaneously about 70% of the time. The *ear-voice-span* (EVS, the delay between hearing and translating) is typically two to six seconds (Gerver, 1997). This delay is supposedly made up from filling a kind of buffer with source language fragments, while the language module is still occupied with other types of tasks (Anderson, 1994). Interpreters may apply different strategies for their task, depending on the structural complexity of the source language and the text to be interpreted, and the degree of structural deviation in the target language. Usually, an interpreter seems to wait until the speaker has uttered a sentence-like fragment. Only then — e.g. when a nominal phrase and a predicate are present — the translation process and production of target language text starts (Goldman-Eisler, 1972). This amounts to a delay until enough syntax and semantics are present to produce coherent text in the target language.

This structure cannot be applicable if the structures of the source and target language are very different. For example, given the translation task from a language that is verb-final into a language that shows the verb in the second position, for reasons of time, it is not feasible to wait for the predicate.[17] Moreover, the pressure on the interpreter increases, as he has to memorize more and more material from the source language. In these cases, interpreters apply the strategy of open sentence planning, which usually allows them to start the translation in a successful manner, even if the association of complements to roles in the sub-categorization frame of the predicate (or even the predicate itself) is not yet known. In order to leave the functional structure of a sentence in the target language open as long as possible, an interpreter may utter a copula; this fulfills the structural restrictions of the target language. The translation of complements may follow this copula (cf. Kitano, 1994, p. 97).

1.3 Incremental Architectures and the Architecture of MILC

None of the systems for automatic interpreting existing today actually carry out simultaneous interpreting, few have this as an explicit goal.[18] In fact, the vast majority

[17] Matsubara and Inagaki (1997b) present a translation system that translates incrementally from English to Japanese. This system utilizes the fact that English is verb-second, and Japanese on the other hand is verb-final. To account for the translation of complements of English verbs, the system first translates a very short sentence into Japanese, solely consisting of the subject and the verb. The complements still missing are presented in a second translated sentence, which is concluded by a repetition of the verb. These sentences have a medium-high stylistic quality, because the second sentence does not repeat the subject (Japanese allows zero subjects) (Matsubara and Inagaki, 1997a; Mima, Iida and Furuse, 1998). This kind of processing is heavily dependent on the individual languages involved. Given this schema, a reversal of the translation direction is not possible.

of systems work non-incrementally.[19] The system presented here does not interpret simultaneously either. However, as incrementality is a fundamental property of human speech understanding and, in our view, is essential for successful, adequately communicating technical systems for the processing of language, the concept of incrementality is the starting point and motivation for the work presented here.

So far, interpreting systems with a broad coverage, e.g. Verbmobil (Wahlster, 1993) and Janus (Waibel, 1996), and other kinds of NLP systems have some architectonic principles in common:

- They are modular. Different software components are responsible for the processing of input on different levels. The modularization that is most commonly used is oriented towards the supposed levels of linguistic processing. In certain cases the distribution of tasks over modules may vary, e.g. in the case of Verbmobil, which contains several modules for translation depending on the translation approach (cf. Section 4.8.1). Each module, however, has a monolithic structure that accounts for the difficulty those systems pose for parallelization.
- They employ many different mechanisms to represent linguistic knowledge and to store and transmit intermediate results. The consequence of this technique is usually a very intensive use of resources (especially main memory) and a need for several interfaces connecting the different components.
- They work non-incrementally. This means that the systems wait until the speaker has finished his or her utterance before starting the actual computation.

Natural language processing systems have made a substantial step towards operating on spontaneous speech in the last few years. Earlier, the predominant input consisted of read or controlled speech (Waibel *et al.*, 1991). As spontaneous speech poses completely new problems on the levels of both speech and language, this transition has a severe influence on the structure and the content of knowledge sources and the components which use them. In addition to the big influence of continuous speech, which basically also shows up in read speech, spontaneous speech adds a new dimension of complexity by introducing acoustic phenomena like hesitations, cutoffs and pauses. In the domain of linguistic processing, fragmented utterances surface frequently, yielding input which is agrammatic if a standard grammar is applied. The use of elliptic utterances increases. Additionally, almost all utterances are embedded into a speaking situation, e.g. in a dialog. Ellipses, anaphoric references and deictic expressions are perfectly understandable for human dialog partners, but pose serious trouble for an artificial analysis (cf. Tropf, 1994).

The system we discuss here (MILC, *M*achine *I*nterpreting with *L*ayered *C*harts) is a prototypical implementation of a completely incremental system for the automatic translation of spontaneously spoken speech. Assuming the principle of incre-

[18] An exception from this rule is ΦDMDIALOG (Kitano, 1990; Kitano, 1994), a highly parallel system which carries out example-based translations.

[19] Incrementality is integrated on different levels, rendering modules incremental, without yielding a complete incremental system (Wirén, 1992; Finkler and Schauder, 1992; Schröder, 1993; Poller, 1994; Kilger, 1994; Milward, 1995; Hanrieder, 1996).

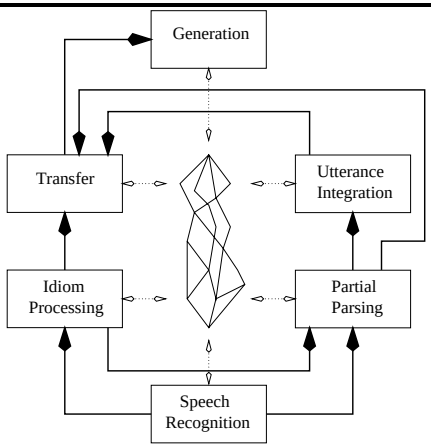


Figure 1.2. The architectonic overview of MILC

mentality, architectonic schemata can be defined that model the information flow within systems grounding on that principle. A prominent representative of such a model is the *cognitively oriented architecture model* by Briscoe (1987), which has been used as a guideline for the ASL-project (cf. Görz, 1993). This model is primarily concerned with the modularization of components and the interactions that are possible between components. Each module is autonomous according to the *weak modularity hypothesis*. Connections between modules come in two flavors. The first constitutes the main data path that follows the linguistic strata. Additionally, there are interaction channels that are used to transmit restrictions for search spaces and predictions to other components (Hahn, 1992; Pyka, 1992c). These additional communication paths can also be used to initiate question-answer sequences, which are useful for disambiguation.[20]

The design of MILC follows this idea of architecture by exhibiting a modular system with dedicated information channels (Amtrup, 1994b; Amtrup and Benra, 1996)(cf. Figure 1.2).[21] During the design and implementation of the system, the principle of *integration* was strictly obeyed. By integration we mean that there is one coherent data structure in the system used to store intermediate results. In the case of MILC, we developed a new data structure for this purpose, called *layered chart* (Amtrup, 1997b; Amtrup, 1997c). Using layered charts simplifies the distribution of results produced by a module to a great extent, since references become valid

[20] Naturally, other architectonic schemata using different principles are conceivable. One example is given by Levelt (1989), who constructs a model of speech production which works almost completely without interaction.

[21] The current implementation does not utilize interaction channels.

across module boundaries. The use of a layered chart is indicated in the center of the components of Figure 1.2. Chapter 4 gives a detailed account of the modules of MILC and the interactions among them. For now, we only want to mention that a layered chart constitutes a graph structure that opens a direct way to inter-modular parallelization. Moreover, at any given time, a consistent view of the system state is possible by simply computing the union of the information present in all modules[22], without demanding that global structures exist (as is the case in the approach of Boitet and Seligman (1994)).

In the framework of an incremental system, MILC not only employs integration, but also the principle of *uniformity*. This stands for the use of a homogeneous formalism, wherever this is reasonably possible (Kay, 1984; Weisweber, 1992). Strictly speaking, this property is not decisive for the abilities and efficiency of any single component. However, using a uniform formalism contributes to a better understanding, extensibility and ability to maintain a system, e.g. by avoiding complex conversions associated with interfaces between modules. We developed a complex typed feature structure formalism (cf. Carpenter, 1992) as it is used in the majority of modern natural language processing systems. This formalism is used in all modules to declaratively formulate and process linguistic knowledge (cf. Emele *et al.*, 1991), ranging from the recognition of idioms to generation. The implementation is oriented on abstract machines that are shift invariant (cf. Carpenter and Qu, 1995; Wintner and Francez, 1995b; Wintner and Francez, 1995a and Section 3.3) and contributes much to the simple fashion of distribution of the system, as well as to the efficient use of inter-modular parallelism.

An important advantage that follows from observing integration and uniformity lies in the possibility to explore the effects of complex interaction patterns between different components. Consider the presence of different analysis paths in a system. Usually, the only way to combine the work of two paths is to collect the final results of both methods and to select the better one. Using an integrated, uniform way of processing, intermediate results can be exchanged between the two lines of processing, thereby facilitating a mutual influence. This type of interaction and influence is demonstrated using the recognition and processing of idioms in MILC (cf. Section 4.5).

The predominant method used in this work to structure the system on all levels is to use graphs as a medium for representation. First, a layered chart is a directed acyclic graph with a distributed storage schema, the graph-theoretic properties of which will be extensively discussed in Chapter 2. Second, the input to the system consists of word graphs. We will define them and the algorithmic means for their manipulation in Section 2.2. Third, the representation of all kinds of intermediate results and also the final representation of the translation is performed using a graph structure. To facilitate the final output, a linear form is generated incrementally from the generation graph. Fourth, the linguistic description associated with each edge is made from typed feature structures, which are directed, labeled, but potentially

[22] The global state of a distributed system is unknown by definition. The view taken here is that there is always a reasonable combination of data items present in different modules.

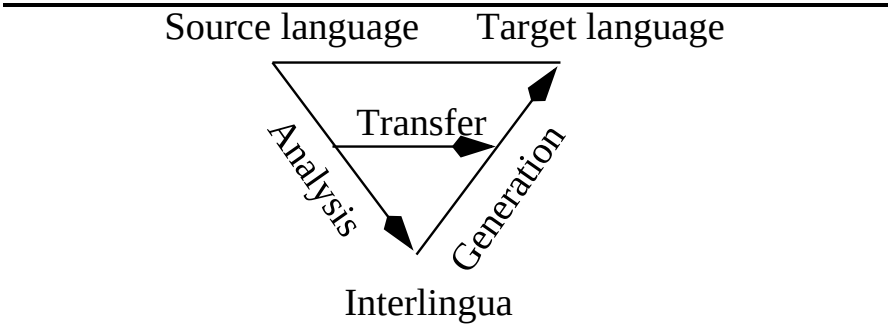


Figure 1.3. Interlingua and transfer

cyclic, graphs. The underlying type lattice is a directed, connected, acyclic, rooted graph (cf. Chapter 3).

Finally, on a higher level of abstraction, MILC itself is a graph of components, the edges of which represent communication paths (cf. Section 4.2). The component graph may be cyclic in order to introduce feedback loops. This property was not used in the current implementation of MILC.

MILC is a translation system following the transfer paradigm. The fundamental difference between transfer systems and their counterpart, interlingua-based systems, lies in the status of the information which is used to mediate between source and target language (Hutchins, 1986). Interlingua systems (Nirenburg, 1987; Nirenburg, 1992) use an internal representation of language expressions, which is independent of the actual language used to express them (cf. Figure 1.3). This implies that one needs $2n$ components to construct a translation system between n languages: One analysis module and one generation module for each language.

Systems that use the transfer paradigm (see, e.g. Nagao and Tsujii, 1986; Luckhardt, 1987; Grabski, 1990) do not have such a united representation. Rather, there is one specialized transfer component for each pair of languages (or, in the extreme, for each translation direction). Although this requires $3n(n-1)$ components in the worst case, transfer systems seem to be more and more accepted. The reason for this is not only the philosophical discussion about the necessary power an interlingua needs to express all subtleties of linguistic and cultural nature (Hutchins and Somers, 1992; Nirenburg, 1993),[23] but also the fact that it is unnecessary in many cases to disambiguate extensively (Zajac, 1990; Somers, 1993).

Using transfer-oriented systems, it is possible to restrict the depth of analysis to a degree sufficient for the translation between a specific pair of languages. In general,

[23] There are even approaches that use English or Esperanto as interlingua in machine translation systems (cf. Schubert, 1992).

this means less effort, e.g. because the cultural relation between the languages in question is known beforehand.

Independent of the type of analysis applied to a certain language, there will be cases in which specific parts of an utterance are analyzed too deeply before transfer tries to create a translation. For each level of linguistic abstraction, there are phenomena the translation of which can only be performed if knowledge from this specific level is used (Kinoshita, Phillips and Tsujii, 1992). To cite only a few:

- *Idioms* and *routine formulae*. The translation of these elements is best established according to lexical information, because they most often show a non-compositional semantics by being relatively firm structurally at the same time.
- *Head switching*. This is a frequently cited example for a structural transfer problem on a syntactic level.[24] The phenomenon is the following: Adjuncts in the source language have to be translated as heads in the target language realization in some cases. Consider **(1.7)**. The German adjunct gerne has to be translated as head of the English phrase (likes).

Hans	*schwimmt*	*gerne*
John	swims	likingly

  "John likes to swim."

  (1.7)

- *Tense* and *aspect*. They can only be correctly translated on the basis of semantic information. In many cases, even extralinguistic knowledge is necessary (Eberle, Kasper and Rohrer, 1992).

Usually, transfer takes place after the syntactic/semantic analysis of input utterances. Any information gathered along with the processing is accumulated monotonically in linguistic objects, thereby allowing the transfer to use results from preceding stages, e.g. from prosody. Therefore, the treatment of the problems stated earlier is possible. But most often there is no method for regulating the depth of processing. Once the decision is made as to which level of representation is going to be the foundation of transfer, it is generally accepted that every input utterance is analyzed up to that level. However, this is not an imperative.

Experienced human interpreters, for example, translate some utterances almost automatically (Hauenschild, 1985). An in-depth analysis of the material heard seems to take place only when the material is linguistically complex or contains ambiguities which are difficult to solve. Hauenschild and Prahl (1994) use this observation to propose the introduction of a *variable transfer*, which would account for different demands during automatic translation (cf. Weisweber and Hauenschild, 1990).

The goal of a variable transfer would be to analyze utterances and their parts only up to the depth which is needed for a successful transfer into the target language. Multi-level transfer according to the principle of a variable depth of analysis demands that there are as many levels of transfer as levels of linguistic processing; consequently, transfer may start at several points during analysis (cf. Figure 1.4).

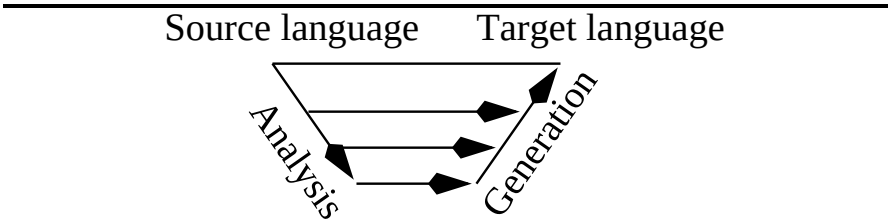


Figure 1.4. Multi-level transfer

The integration and uniformity of MILC provides an optimal architectonic basis for the exploration of different types of transfer mechanisms. We show an example of this by implementing the transfer of fixed idiomatic expressions, which are recognized by a specialized component. Their non-compositional analysis is directly delivered to the transfer component. This schema does not only avoid redundancy, but also adds efficiency, as the recognition of idioms is far less complex than a full-fledged linguistic analysis.

Computing time plays a big, probably a decisive, role especially for applications that use spoken language as input or output modality. The translation of a written text can be performed during nighttime without severe negative consequences. The threshold of tolerance while waiting for the response of an interactive system (in the sense of a dialog system), on the other hand, is in the range of several seconds only. For this reason, the development of *anytime algorithms* (Russel and Zilberstein, 1991; Mouaddib and Zilberstein, 1995) for natural language processing systems has been demanded quite early (Wahlster, 1993; Menzel, 1994). The ultimate goal is, naturally, the construction of a system translating spoken language synchronously. This is, of course, illusionary at present. However, the least one could demand from a system could be some sort of awareness about the limited amount of time that can be used for processing. Alternatively, one could demand that a system be interruptable by a user.

MILC does not implement anytime algorithms. Later, we will discuss the interrelations between anytime algorithms, complex formalisms, modularity and incrementality in detail (cf. Section 4.11.2); right now we only need to say that, again, incremental behavior is a prerequisite for the application of anytime algorithms in automatic interpreting systems.

The development of MILC can not be seen isolated from the presence of other systems. During the past several years a number of highly interesting, advanced applications for the translation of written text or spoken language have been designed.

[24] Although Dorr (1993) treats such *mismatches* on a lexical level.

We will only discuss four of them which are relevant for this work: SLT, Trains, TDMT and Verbmobil.

The *Spoken Language Translator* (SLT) constitutes the research environment for translation systems developed by SRI, Cambridge. The initial system used as a starting point for further investigation was composed out of already available, individual components and was used for translation of utterances in the ATIS-domain (Air Travel Information System) (Agnäs *et al.*, 1994). The languages involved from the beginning were Swedish and English; in the meantime SRI has added modules for processing French. Moreover, the isolated character of single modules has been turned into a more integrated system architecture.

The input to the system consists of a list of the five best recognition results, delivered from a word recognizer for each utterance. These five chains are transformed into a small word graph. Two instances of the *core language engine*, a unification-based linguistic analyzer, are used for analysis and generation. The translation is transfer-based and uses a deep linguistic analysis as well as a shallow representation which is oriented at the surface form (Rayner and Carter, 1997). Although the SLT as a whole does not work incrementally, the best translation, resulting from a combination of deep and shallow analysis, is permanently available.

One method to increase the quality of analysis and the efficiency of the overall processing is to specialize a fairly general grammar through domain-specific knowledge. The ambiguity introduced by multiple lexical entries based on syntactic categories can be greatly reduced by using a language model. A supervised training algorithm further eliminates the ambiguity on the constituent level. To facilitate this, so-called discriminants are extracted from constituents, which contain a short description of the constituents in question. These are again used to train a language model that is used as an additional restriction during analysis. Moreover, the SLT exploits the fact that the frequency of the use of certain grammar rules depends on the domain. The general grammar is reduced to the set of rules relevant for the current domain, which results in a significant increase in efficiency by only mildly over-generating the possible analyses (Rayner and Carter, 1996). SLT thus demonstrates a successful integration of symbolic and statistical methods for natural language processing.

Trains (Allen *et al.*, 1994) is not a translation system, but a dialog system capable of performing simple route planning tasks. However, one aspect is interesting for the work described here: The correction of speech recognition errors. Trains uses a best chain recognizer and is thus prone to errors occurring during the initial recognition phase. The solution applied consists of a postprocessing step to correct typical errors that occur most often. The recognition output undergoes certain modifications, the result of which is ranked according to a language model. The modifications consist, in the most part, in the exchange of single words of the lexicon (Allen *et al.*, 1996).

TDMT (*Transfer-Driven Machine Translation*, Furuse, 1994) uses the transfer stage as the central part of processing. A transfer module that is oriented toward previously seen translations receives the surface representation that flows into the system. The transfer rules consist of patterns that relate frequent combinations of

words and phrases (Sobashima *et al.*, 1994). The patterns are used as a trigger for evaluation procedures that may integrate knowledge from several linguistic levels, e.g. lexical knowledge to treat compositional lexical items or syntactic knowledge to handle combinations on the phrasal level. The nature of the transfer knowledge table makes TDMT well suited for parallel processing of translation by partitioning the set of transfer rules into smaller pieces (Oi *et al.*, 1994). By introducing boundary markers it is even possible to introduce an incremental way of processing individual utterances (Furuse and Iida, 1996). However, the setting of boundaries initially relies on the annotation of input words with syntactic categories in instantiate the pattern-based rules.

Verbmobil (Kay, Gawron and Norvig, 1991; Wahlster, 1993), finally, is a major source of inspiration for the research described in this book. The author was member of the project for several years. Verbmobil is an extremely large project for the translation of spontaneously spoken dialogs in the domain of appointment scheduling, consisting of a high number of different modules. It processes German, English and Japanese, but relies on the fact that the dialog participants have at least a passive knowledge of English. The input consists of word graphs which are initially augmented by prosodic information. The exploitation of such knowledge at a large scale has been integrated successfully for the first time. The first phase of the project (1993-1996) processed the word graphs using a deep linguistic analysis and independently used two different types of shallow translation methods based on surface representations and statistical information. Meanwhile, in the second phase, the different approaches for analysis and transfer are combined in a holistic fashion to be able to exchange useful information, even within an utterance (Worm, 1998). The modeling of dialog knowledge and parts of the transfer mechanism are oriented towards dialog acts (Jekat *et al.*, 1995). Verbmobil reaches a performance of more than 70% approximately correct translations (Wahlster, 1997).

Verbmobil as a whole does not realize an incremental system throughout (Hahn and Amtrup, 1996), although some of the components and algorithms use incremental methods (e.g., Finkler, 1996). In particular, the input to the system consists of a word graph which was constructed non-incrementally. Parts of the linguistic analysis is based on a non-incremental A*-search (Caspari and Schmid, 1994). However, the experimental translation system *INTARC* (*INT*eractive *ARC*hitecture), which has been developed in the architecture subproject of Verbmobil, indeed realizes an incremental approach to speech translation (Görz *et al.*, 1996). There is one important restriction for the amount of incrementality that can be experienced in the system: The translation can only be performed for sentence-like constituents that start at the onset of an utterance. An analysis of fragments has not been integrated. The main properties of INTARC are:

- Word graphs are produced incrementally (Huebener, Jost and Heine, 1996). They are processed strictly from left to right.
- Unlike many other approaches, the recognition of prosodic information happens without recourse to the prior presence of recognized words (Strom and Widera, 1996).

- The syntactic-semantic analysis is divided into an incremental search for utterance hypotheses based on the type skeleton of a complex feature structure grammar and the following verification based on the full unification grammar (Kasper *et al.*, 1996).
- The main stream of translation uses a dialog-act based mode of operation. In case of a failure of the deep linguistic analysis there is a fall-back which uses information about focused words in the input utterance and their associated dialog acts for a rudimentary translation (Elsner and Klein, 1996).
- It has been used to investigate interaction between modules for linguistic analysis and speech recognition (Hauenstein and Weber, 1994).

1.4 Summary

Using incrementality within natural language processing systems seems to be the next logical step in the development of large, naturally operating systems. The MILC system, which is described in this book, is an implementation of an incremental system for the translation of spontaneously spoken language. Incrementality can be paraphrased as "processing in a piece-meal fashion", which e.g. renders the effect that a system is able to react to the user's input, even before that input has been completed. Such an operation is oriented towards the properties of human language understanding. It opens the way for new methods to reduce ambiguity (thereby increasing the efficiency) by using interaction between modules and the exploitation of inter-modular parallelism. Incremental strategies are used by simultaneous interpreters, among other methods, to reduce the burden of processing with an open sentence planning.

MILC is a modular system. The components are interconnected using a communication system that operates with channels. The components form an integrated architecture by using a new data structure, the layered chart, to store results and intermediate partial analyses. A complex typed feature structure formalism is used to represent linguistic knowledge on all levels. It is implemented using abstract automata techniques and guarantees a simple, efficient exchange of feature structures among modules. The mode of operation has been demonstrated using five dialogs in the domain of appointment scheduling, which were translated approximately correct to 60.4%.

Chapter 2

Graph Theory and Natural Language Processing

Graph theory is one of the central foundations for this work. All relevant data structures and mechanisms can be traced back to operations over graphs. Therefore, this chapter will present some concepts from graph theory, which are important for the definition and discussion of algorithms for speech processing. We begin with general definitions used throughout this monograph and finish with incremental graph search algorithms.

Graphs are fundamental to natural language processing. Within the framework of the research reported in this book, at least four aspects have to be considered: First, all algorithms used for the linguistic treatment of an input utterance are based on the notion of a chart, which is a special type of graph. Second, the input used for our system consists of word graphs produced by a speech recognizer. Third, annotations on edges are feature structures, which are graph-shaped. And finally, the application itself is graph-structured with vertices made from components and edges made from communication paths.

2.1 General Definitions

We begin by defining the general notion of a *graph* and its components, *vertices* and *edges*. More thorough introductions to graph theory can be found in Chen (1971) or Harary (1974); here we concentrate on the concepts relevant for our purpose.

Definition 2.1.1 (Graph).
A graph G is a pair $G(\mathcal{V}, \mathcal{E})$, which consists of a finite set $\mathcal{V}$ of vertices (nodes) and a finite set $\mathcal{E}$ of unordered pairs (v, v'), $v, v' \in \mathcal{V}$, which are called edges.

The graph $G(\emptyset, \emptyset)$ is called *empty graph*. The edges in this definition have no direction, the endpoints v and v' of an edge (v, v') have equal status. Loops, i.e. edges of the form (v, v), which build connections between one and the same vertex, are explicitly allowed.

In this work we will almost exclusively use graphs within which the direction of an edge is as important as the identity of the vertices that are connected by an edge. The main reason for this lies in the fact that linearity is one of the characteristic properties of language. The lowest level of description for input utterances is

J.W. Amtrup: Incremental Speech Translation, LNAI 1735, pp. 25-63, 1999.

given by chronologically continuous sets of word hypotheses. Linguistic descriptions always range over intervals in time of an utterance. Additionally, the formal description of linguistic knowledge (see Chapter 3) is built on graphs with directed edges in order to represent hierarchical contexts. Thus, in the following we will always deal with *directed graphs* and call them graphs for brevity.

Definition 2.1.2 (Directed graph, associated graph).
A directed graph G is a pair $G(\mathcal{V}, \mathcal{E})$, which consists of a finite set $\mathcal{V}$ of vertices and a finite set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of pairs of vertices, which are called directed edges (written (v, v') or vv'). The undirected graph, which results from neglecting the direction of edges, is called the graph associated to G.

In contrast to undirected graphs, $\forall_{v,v'} : vv' \in G \implies v'v \in G$ does not hold in general for directed graphs. Often, it is useful to allow more than one edge between two vertices. In the field of natural language processing, this can be used e.g. to represent ambiguity or knowledge of different origins for a portion of the input. Consequently, we introduce an indexing of edges in order to make different edges that connect identical pairs of vertices distinguishable. However, we want to represent more information about an edge than a simple natural number. Thus, we use *edge labels*, which can be arbitrary objects. In particular, we use words which represent parts of the input utterance, and feature structures, which represent linguistic hypotheses about segments of the input.

Definition 2.1.3 (Labeled graph).
A labeled graph G is a triple $G(\mathcal{V}, \mathcal{E}, \mathcal{L})$, which consists of a finite set $\mathcal{V}$ of vertices, a set $\mathcal{L}$ of labels, and a finite set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{L}$ of edges, each of which carries a label.

Additionally, we can associate each edge with a *weight*, for example the acoustic score as a measure of how well an interval of the signal matches with the model of a word in the recognizer dictionary. Edge weights, in particular, are useful to rank the relevancy an edge can possibly have for the course of processing. Using this information, the order in which certain processing steps are carried out can be determined.

Definition 2.1.4 (Weighted graph).
A weighted graph G is a triple $G(\mathcal{V}, \mathcal{E}, \mathcal{W})$, consisting of a finite set $\mathcal{V}$ of vertices, a set $\mathcal{W}$ of possible weights and a finite set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{W}$ of weighted edges. Usually, weights are taken from $\mathbb{R}$.

In order to access the components of a graph and its individual parts, we introduce some utility functions:

Definition 2.1.5 (Access functions for graphs).
Let $G(\mathcal{V}, \mathcal{E}, \mathcal{W}, \mathcal{L})$ be a directed graph with edge labels and edge weights. Let e be an edge of this graph with $e = (v, v', w, l)$. We define

- *α as access function to the start vertex of an edge*

$$\alpha : \mathcal{E} \longrightarrow \mathcal{V}, \alpha(e) := v \tag{2.1}$$

- *β as access function to the end vertex of an edge*

$$\beta : \mathcal{E} \longrightarrow \mathcal{V}, \alpha(e) := v' \tag{2.2}$$

- *l as access function to the label of an edge*

$$l : \mathcal{E} \longrightarrow \mathcal{L}, l(e) := l \tag{2.3}$$

- *w as access function to the weight of an edge*

$$w : \mathcal{E} \longrightarrow \mathcal{W}, w(e) := w \tag{2.4}$$

Complete linguistic analyses always describe a traversal of the graph representing the input. Thus, we need a notion of a *path* through a graph or parts of it. Additionally, we introduce the concept of *reachability*, which states whether there is a way to reach one vertex from another one by following a sequence of edges.

Definition 2.1.6 (Adjacency, Reachability).
Two vertices v and v' of a graph $G(\mathcal{V}, \mathcal{E})$ are called adjacent ($v \rightarrow v'$), if $vv' \in \mathcal{E}$:

$$\forall v, v' \in \mathcal{V} : v \rightarrow v' \Longleftrightarrow \exists e \in \mathcal{E} : \alpha(e) = v \wedge \beta(e) = v' \tag{2.5}$$

The transitive hull of the adjacency relation $\rightarrow$ is called reachability relation, written $\overset{*}{\rightarrow}$.

Similarly, we call two edges e, e' of that graph adjacent, written $e \rightarrow e'$, iff the end vertex of e is the start vertex of e':

$$\forall e, e' \in \mathcal{E} : e \rightarrow e' \Longleftrightarrow \beta(e) = \alpha(e') \tag{2.6}$$

Definition 2.1.7 (Edge sequence, Edge train, Path, Cycle).
Let $G(\mathcal{V}, \mathcal{E})$ be a graph, $e_{[1;n]}$ a finite series of edges of the graph with the length n. We write $e^{(i)}_{[1;n]}$ to denote the i-th component of this series, $i \in \{1 \ldots n\}$.

The series is called edge sequence, iff all neighboring edges are adjacent, i.e. iff:

$$\forall_{i=1\ldots n-1} \beta(e^{(i)}_{[1;n]}) = \alpha(e^{(i+1)}_{[1;n]}) \tag{2.7}$$

An edge sequence is called closed, if $\alpha(e^{(1)}_{[1;n]}) = \beta(e^{(n)}_{[1;n]})$, else it is called open. An edge sequence is an edge train if the edges of the sequence are pairwise different, so that

$$\forall_{i,j \in \{1\ldots n\}} e^{(i)}_{[1;n]} = e^{(j)}_{[1;n]} \Longrightarrow i = j \tag{2.8}$$

An open edge sequence is called path if no vertex occurs twice in the sequence, i.e.

$$\forall_{i,j \in \{1\ldots n\}} \beta(e^{(i)}_{[1;n]}) = \alpha(e^{(j)}_{[1;n]}) \Longrightarrow j = i + 1 \tag{2.9}$$

A closed edge sequence is a cycle, if no vertex except $\alpha(e^{(1)}_{[1;n]}) = \beta(e^{(n)}_{[1;n]})$ occurs more than once.

Definition 2.1.8 (Length of an edge sequence, Distance).
The length of an edge sequence is defined as the number of edges in it. The distance $\delta(v, v')$ *of two vertices is defined as the length of the shortest path between them. If there is no such path, we set* $\delta(v, v') = \infty$.

Two other fundamental concepts of graph theory, *connectivity* and *planarity*, are only of minor interest for this monograph. We will implicitly request graphs to be connected in later definitions, hence unconnected graphs do not arise. Also, the graphs used in our research are usually so dense that they are not planar.

Definition 2.1.9 (Connectivity).
A graph is connected if any two vertices are reachable by following a path through the graph. A directed graph is called strongly connected if any vertex is reachable from any other vertex by a path. It is called half connected if there is a path between any two vertices. The graph is called loosely connected if the associated undirected graph is connected.

Definition 2.1.10 (Planarity).
A graph is planar if it can be drawn on a surface without crossing edges.

Charts, which are used for natural language processing, are in general not planar. Let us assume a German sentence as an example:

Der	*Peter*	*singt*	*mit*	*Freude.*
The	Peter	sings	with	joy.

Peter sings happily. **(2.1)**

The syntactic analysis using usual methods creates a chart similar to that in Figure 2.1. It contains at least pre-terminal edges resulting from lexical access, and some inactive edges representing different sentential hypotheses and the complete verbal phrase. This chart is isomorphic to the graph $K_{(3,3)}$ in Figure 2.2. This is a complete bipartite graph with six vertices, which are partitioned in two sets of three vertices each. Following the theorem of Kuratowski (Aigner, 1984, S. 74), the graph $K_{(3,3)}$, and consequently the chart shown here, are not planar.

Usually, the proof of non-planarity can be made easier by using a theorem about the restriction on the linearity of the number of edges in a graph (cf. McHugh, 1990, S. 38), which goes back to Euler's formula. Using this theorem, it is especially easy to show the non-planarity of word graphs.

The notion of a chart[1], used for a data structure which stores complete and incomplete hypotheses over a certain extract from the input, has consequences for the shape and properties of the graph representing it. However, this depends on the application in which the chart is used. If written input is considered, the vertices

[1] We will define a chart more formally in Definition 4.1.1.

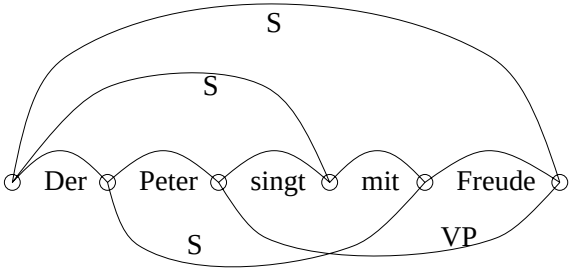


Figure 2.1. A chart for Der Peter singt mit Freude

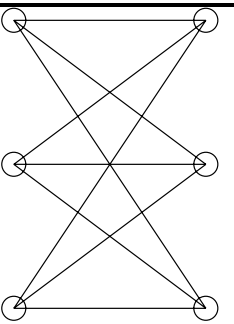


Figure 2.2. The graph $K_{(3,3)}$

of the graph represent the gaps between words.[2] Each two neighboring vertices are connected by (at least) one edge, which denotes the current input word. Thus, the graph is half connected, the distance between two arbitrary vertices is finite ($\forall_{v,v' \in V} : \delta(v, v') \neq \infty$). The reachability relation is a total ordering on the vertices of the graph.

If a chart is used to represent structures that allow alternatives on the lowest level (in general words), this is no longer the case. The output of a word recognizer could consist of a word graph containing multiple alternative paths. This means that vertices are constructed that are no longer connected by a path whose distance is thus ∞. The reachability relation is only a half ordering. All paths from the start vertex of the graph to the end vertex of the graph are half connected graphs, they represent alternative utterance hypotheses.[3]

The following definition of a *directed, acyclic graph* is important for our purpose, because almost all graphs used in computational linguistics have this property. Charts and word graphs are directed, yet they have an additional property, namely having a linear character: They do not contain edges that reach backwards in time. They are even acyclic.[4]

Definition 2.1.11 (Directed, acyclic graph (DAG)).
A directed, acyclic graph (DAG) is a directed graph which does not contain directed cycles.

[2] Boitet and Seligman (1994) use the name chart exclusively for this case. We use a more general definition (see Section 4.1).

[3] Later (cf. Section 2.6) we will introduce a natural total ordering on vertices, which rests on the projection onto the time axis.

[4] The syntactic analysis using a chart sometimes utilizes empty active edges, which do not lead backwards in time, but nevertheless introduce cycles. We abandon these by using a left-corner analysis following Kilbury (1985) (cf. Chapter 4).

2.2 The Use of Word Graphs for Natural Language Processing Systems

Systems using written language as input are well served by having no doubts about the identity of the input.[5] If, however, the input consists of spoken language, a whole new set of problems arises, which on one hand results from the properties inherent to speech, and on the other hand is caused by the methods available for speech recognition.

- *Recording conditions.* The most prevalent source of a suboptimal acoustics is the presence of disturbing sound like background noise, etc. As long as the noise is more or less static, it can be removed fairly easily. Non-stationary noise like door slamming, radio broadcasts, telephones ringing, etc. are harder to detect and suppress.
- *Incomplete recognition*. In certain cases there is no guarantee that the input level is constant over time. This may be caused by movements of the speaker in relation to the microphone. Suitable recognition situations (e.g. wearing head sets) can help minimize these problems.
- *Speaker variation*. Aside form the almost classical difference in recognition quality between voices of men and women, dialect and variation of speaking speed may affect the recognition considerably.
- *Connectivity*. The lack of clearly defined and distinguishable word separations during the recognition of continuous speech leads to an explosion of the size of the search space which has to be traversed by the recognizer. However, using a single-word recognizer, which shows much higher accuracy, is out of the question for the research presented here.
- *Performance phenomena*. Break offs, hesitations and speaker noise are the most common performance phenomena. They occur extremely frequently in spontaneous speech, but are rare in e.g. read speech.
- *Mathematical foundations*. Modern speech recognizers are almost always based on stochastics and HMMs[6]. HMMs are finite state transducers, where transitions and emissions are modeled using distributions. These distributions are learned on the basis of a large set of training data. The selection and coverage of the training data determines to a large degree the performance of the recognition step when processing previously unseen input.

Roughly said, you can never be sure that a recognizer does recognize what has been said. The form of the word recognition output reflects this situation. In principle, three interfaces are used to connect a recognizer with linguistic modules:

The simplest interface is to use the **best chain**. In this case, the recognizer delivers a sequence of words which was attributed with the highest available acoustic

[5] But cf. Wirén (1992) for an account on the modeling of undoing words by correction of typos.

[6] *H*idden *M*arkov *M*odel, cf. e.g. Woodland *et al.* (1995).

score.[7] The main advantage of this approach lies in the fact that modules capable of processing written input can be used without modification to accept spoken language as input. Moreover, current recognizers are able to deliver word recognition rates of 90% and more if the lexicon is small enough. The two main disadvantages are the poor scalability to large vocabularies and the low sentence recognition rate: Although nearly all words are recognized correctly, almost no sentence is recognized completely.

An easy modification consists of recognizing many instead of just one sentence hypothesis (**n-best hypotheses**) (cf. e.g. Tran, Seide and Steinbiss, 1996). The hope is that the probability of finding the right utterance is higher if ten or twenty candidates are available. While this is certainly true, it presents only a marginal augmentation, since the number of potential utterance hypotheses that are stored inside the recognizer is larger by orders of magnitude than the best n chains. Moreover, the isolated processing of a number of hypotheses which most often differ in only one word seems hardly adequate.

Word graphs (Oerder and Ney, 1993; Aubert and Ney, 1995) are the mode of representing word recognition results, which have been used more and more during the past years. Word graphs are directed, acyclic graphs, whose edge labels denote words in orthographic transcription. Additionally, edge weights represent acoustic scores[8] for word hypotheses.[9] The graph contains unique start and end vertices which represent the boundaries of the input utterance. Each path from the start vertex to the final vertex describes a possible utterance hypothesis. Each vertex within the graph denotes a point in time during the speaking time.

In order to formally define word graphs, we need the notion of the degree of a vertex:

Definition 2.2.1 (Incidence, Degree of a vertex).
Let $G(\mathcal{V}, \mathcal{E})$ be a graph. An edge $vv' \in \mathcal{E}$ is called incident from v and incident to v'. For a vertex v of the graph G, the in-degree is defined as number of edges incident to v:

$$\#_{in}() : \mathcal{V} \longrightarrow \mathbb{N}, \#_{in}(v) := \#\{e \in \mathcal{E} | \beta(e) = v\} \tag{2.10}$$

The out-degree of a vertex v is defined as the number of edges incident from v:

$$\#_{out}() : \mathcal{V} \longrightarrow \mathbb{N}, \#_{out}(v) := \#\{e \in \mathcal{E} | \alpha(e) = v\} \tag{2.11}$$

[7] I.e., the sequence of words that belongs to the set of training data which best fits the actual input.

[8] Usually, scores are taken as negative logarithms of densities. Thus, the lower the weight, the more confident the recognizer has been about the presence of a word in the input signal.

[9] In order to improve the recognition rate almost all word recognizers utilize statistical language models, which consist of transition probabilities between words. Using these figures, the number of words which may follow a certain word is reduced. It is not possible to represent these transition probabilities in the resulting word graph, although they are used for pruning in the recognizer. However, modules for linguistic analysis may again use language model probabilities for their own purposes.

Definition 2.2.2 (Word graph).
A word graph is a directed, acyclic graph $G(\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{W})$ with edge labels and edge weights, if the following holds:

- *The graph contains an unique start vertex. This vertex represents the start point of the utterance.*

$$\exists v^{(r)} \in \mathcal{V} : \#_{in}(v^{(r)}) = 0 \wedge \forall v \in \mathcal{V} : \#_{in}(v) = 0 \Longrightarrow v = v^{(r)} \quad (2.12)$$

  From the acyclicity and the existence of $v^{(r)}$ it already follows that the graph is loosely connected.
- *The graph contains an unique end vertex. This vertex represents the endpoint of the utterance.*

$$\exists v^{(f)} \in \mathcal{V} : \#_{out}(v^{(f)}) = 0 \wedge \forall v \in \mathcal{V} : \#_{out}(v) = 0 \Longrightarrow v = v^{(f)} \quad (2.13)$$

Additionally, each vertex is labeled with a point in time, which can be used to impose an order among the vertices (the linear order in time). Time is discretely represented in intervals of 10ms, the so-called frames.[10] *To do this, we define a function that maps vertices to frame numbers:*

$$t : \mathcal{V} \longrightarrow \mathbb{N} \quad (2.14)$$

t is not injective, because we cannot guarantee the existence of a vertex for every frame. Neither is t surjective, since there may be several vertices for a given point in time, depending on the circumstances.

Edge labels denote words, edge weights denote acoustic scores.

An example for a typical word graph is shown in Figure 2.3. The reader should note, however, that this graph was constructed non-incrementally. This is easy to see: An incrementally produced word graph would contain many "dead ends" which are constructed if a path cannot be further pursued from a certain point, because no additional word hypotheses with a sufficiently high confidence could be found.[11] To describe incremental graphs correctly, proposition (2.13) has to be modified. There may be arbitrary many end vertices (vertices with out-degree zero). After processing, only those paths through the graph which start at the start vertex and end at the rightmost end vertex represent complete utterance analyses.

This property is captured in the following definition which describes incrementally produced word graphs (called left-connected word graphs in Weber, 1995).

[10] This notion stems from speech signal processing. During a preprocessing step, the intensity values are mapped to feature vectors. This method uses windows over the input signal, which are moved in 10ms steps.

[11] The visualization of incremental word graphs is useless for the most part. The fan out is normally too high, producing only a wide, blackened area on a printout (but cf. Section 4.10).

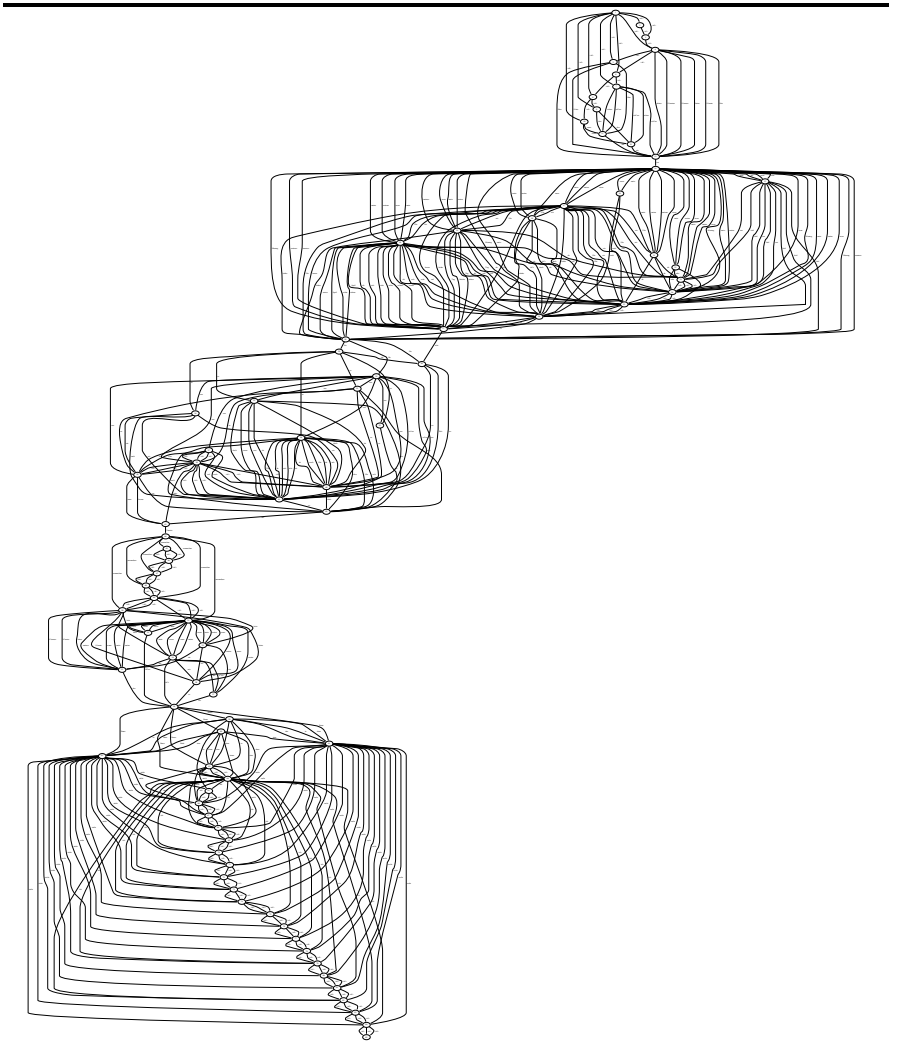


Figure 2.3. Word graph for the utterance n002k000: "schön hervorragend dann lassen Sie uns doch noch einen Termin ausmachen wann wäre es Ihnen denn recht". This graph shows the non-incremental output of a word recognizer.

Definition 2.2.3 (Left-connected (incremental) word graph).
A left-connected word graph is a directed, acyclic graph with edge labels and edge weights, which contains an unique start vertex.

Sharing partial paths in a word graph (by having identical prefixes or postfixes) leads to a highly compact representation of a very large number of alternative utterance hypotheses. For example, the graph for the turn n002k000 presented in Figure 2.3 contains only 461 edges, but describes $1,2 \cdot 10^{23}$ different paths. However, even word graphs contain many redundancies by having several different paths labeled with identical word sequences. This may happen by having two edges emerging from the same vertex and describing the same word that end in different vertices (e.g., at different points in time). The aforementioned word graph only contains $8,6 \cdot 10^{8}$ distinct word sequences. The graph of these sequences is shown in Figure 2.4[12].

An important property is the possibility to impose a topological ordering on the set of vertices.

Definition 2.2.4 (Topological order of a DAG).
The topological order of a directed, acyclic graph $G(\mathcal{V}, \mathcal{E})$ is a mapping $\tau : \mathcal{V} \longrightarrow \mathbb{N}$ with

$$\forall e \in \mathcal{E} : \tau(\alpha(e)) < \tau(\beta(e)) \tag{2.15}$$

The topological order of a DAG can be computed in $O(|\mathcal{V}| + |\mathcal{E}|)$ time as described by Algorithm 1. The algorithm presented here is a little bit simpler than in the general case (cf. Cormen, Leiserson and Rivest, 1990, p. 486), because word graphs are loosely connected. The topological order of nodes is used to reduce the time complexity for a number of algorithms shown later.

The next sections are concerned with general properties of word graphs and methods for the reduction of their size. For this purpose, we will not use left-connected graphs, which will be the topic of later sections.

2.3 Evaluation of Word Graphs: Size and Quality Measures

The introduction of word graphs into natural language processing opened up new areas of research topics, especially concerning the evaluation of different methods and systems. Specifically in the field of speech recognition, quantitative evaluation plays an eminent role, as a high recognition rate usually points to a good system. Calculating this measure is relatively simple if the output of a recognizer simply consists

[12] The algorithm to reduce word graphs is described below.

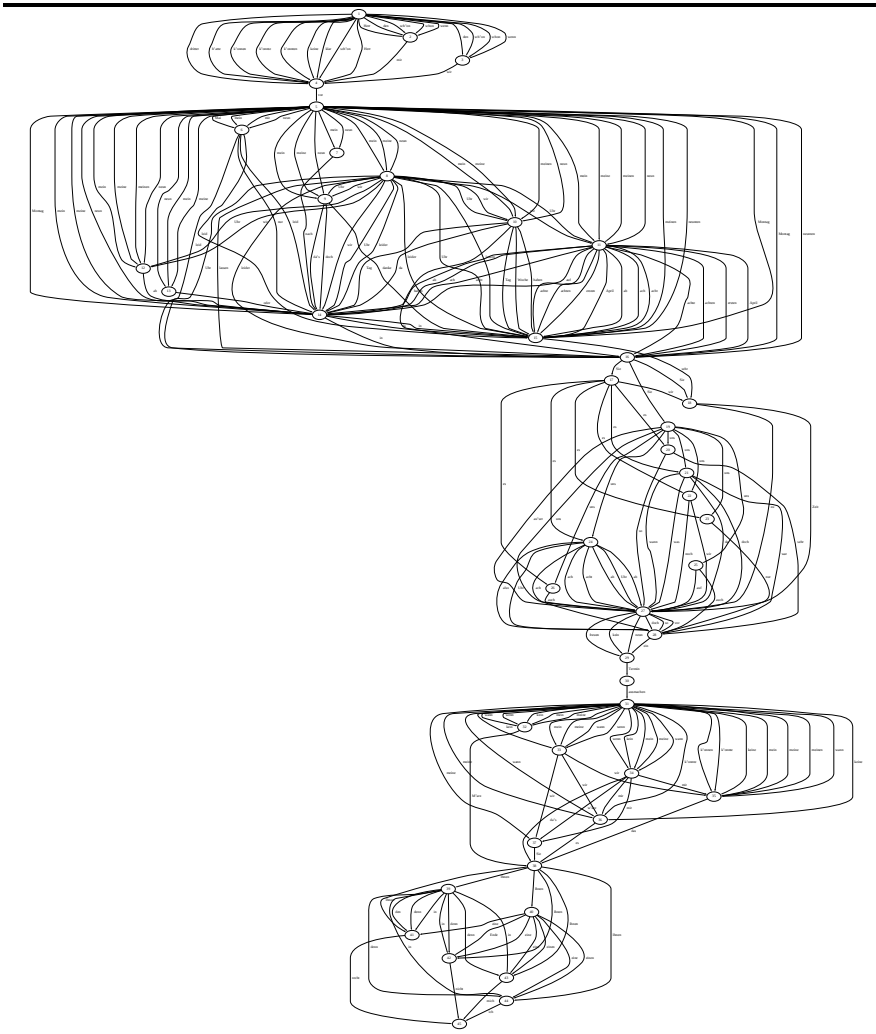


Figure 2.4. Word graph for the utterance n002k000. This graph only contains unique label sequences.

```
begin
[1]     Stack s;
        Initialization
[2]     for each vertex v ∈ V(G), do
[3]         inE_v := #_in(v)
[4]     s.Push( v^(r))
[5]     order ← nil

        Simplified Depth First Search
[6]     while v ← s.Pop() do
[7]         order ← order.v
[8]         for each edge e = (v, v') starting in v do
[9]             inE_v' ← inE_v' − 1
[10]                if inE_v' == 0 then
[11]                    s.Push( v')

[12]    return order
end
```

Algorithm 1. Computing the topological order of a DAG

of a sequence of words. However, if word graphs are considered, their topology cannot be neglected completely. For instance, if the two graphs from Figures 2.3 and 2.4 are considered, it can be noted that the "complexity" is not distributed isomorphically over the whole graph. Rather, there seem to be areas in the graph which obviously contain a high degree of ambiguity, while other areas are relatively simply structured. Especially the "slim" regions where almost all paths collapse to a small number of edges are relevant. In this section, we will review size measures for word graphs and propose a new measure which is independent from transcriptions and, in our view, better takes into account the embedding of a speech recognition system into a larger speech understanding system. In addition, we will briefly discuss quality measures, portraying several different alternatives.

According to the type and form of word graphs, we can state different demands, which unfortunately almost exclude each other:

- The word graph should be small, enabling a linguistic analysis to deliver results in short amounts of time. In the extreme case, this corresponds to using a best chain representation, as has been done for a long time.
- The word graph should be large, thereby containing the actual utterance with a sufficiently high probability. For statistical reasons alone, a higher number of utterance hypotheses results in a higher word recognition rate.

Until now, size measures for word graphs are usually related to the number of edges, for example the density which is defined as number of edges per word in a

begin
Initialization
[1] dens $\leftarrow$ 0
[2] totalDens $\leftarrow$ 0

Consider each vertex
[3] **for** each vertex $v \in \mathcal{V}$, taken in topological order, **do**
[4] totalDens $\leftarrow$ totalDens + dens
[5] dens $\leftarrow$ dens - $\#_{in}(v)$ + $\#_{out}(v)$

[6] **return** totalDens / $(|\mathcal{V}| - 1)$
end

Algorithm 2. Computation of the transcript independent density of a word graph

reference transcription. Let us first additionally define a density which is independent from a transcript:

Definition 2.3.1 (Density, Transcript-independent density).
The (transcript dependent) density of a word graph is the quotient of the number of word hypotheses in it and the number of words present in a reference transcription.

The transcript independent density of a word graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{W}, \mathcal{L})$ is the mean number of edges spanning a certain point in time.

This density can be computed using Algorithm 2 in $O(|\mathcal{V}|)$ time, presumed that the vertices of the graph are ordered topologically.

The most widely used quality measure for word recognition results is *word accuracy*.

Definition 2.3.2 (Word accuracy).
Consider a sequence of words delivered by a word recognizer and a reference transcription of the utterance spoken. The error measure of word accuracy is computed as follows:

$$\textit{Word accuracy} \quad = \quad 100 - 100 \times \frac{\#\textit{Errors}}{\#\textit{Reference words}} \tag{2.16}$$

In this equation, replacements, insertions and deletions are considered to be errors:

$$\#\textit{Errors} \quad = \quad \#\textit{Replacements} + \#\textit{Insertions} + \#\textit{Deletions} \tag{2.17}$$

To transfer the word accuracy from the evaluation of word sequences to the measuring of recognition results on graphs is problematic at best. Usually, the result would be the maximum of the word accuracies computed from all paths in the word

begin
Handle each vertex
[1] **for** each vertex $v \in \mathcal{V}(G)$, taken in topological order, **do**

Compute the number of paths ending at v
$p^{(v)} := \sum p^{(w)} : wv \in \mathcal{E}$
[2] $p[v] \longleftarrow 0$
[3] **for** each edge $e = wv$ ending in v **do**
[4] $p[v] \longleftarrow p[v] + p[w]$
The number of paths in the graph is the number of paths up to the final vertex
$p^{(G)} \longleftarrow p^{(v^{(f)})}$
[5] **return** $p^{(G)}$
end

Algorithm 3. Calculation of the number of paths in a graph

graph. This is done by applying a dynamic programming method, searching for the path through the graph which results in the least number of errors. In order to account for different sizes of word graphs, the accuracy is charted as a function of the (transcript-dependent) density.

For the most part, this procedure neglects the fact that recognition with word graphs per se is only meaningful in connection with a linguistic analysis which follows the recognition phase. This restriction is not necessarily valid with best chain recognizers, e.g. in the case of dictation applications, whose processing stops after recognition. However, for recognition with graphs we feel that an isolated evaluation is only of a lesser importance (cf. Amtrup, Heine and Jost (1996); Amtrup, Heine and Jost (1997)).

A more realistic evaluation takes into account the behavior of modules carrying out a linguistic analysis of word graphs. The first step in establishing a realistic measure for word graph size is to use the number of paths in a graph as this measure. This is based on the assumption that a linguistic analysis (in the following we will always assume a hypothetical parser) might process every path that is contained in the graph.

The number of paths of a word graph grows exponentially with the number of vertices in it (Oerder and Ney, 1993); if the edges are evenly distributed, a graph contains $(\frac{|\mathcal{E}|}{|\mathcal{V}|})^{|\mathcal{V}|-1}$ paths. Algorithm 3 computes the number of paths in an actual graph, $p^{(G)}$. The complexity of the algorithm is $O(|\mathcal{E}| + |\mathcal{V}|)$, presuming the graph is sorted topologically. Although it uses two loops for the computation, the outer loop (line [1]) only determines the order in which edges are considered. No edge is processed twice.

begin
[1] **for** each vertex $v \in \mathcal{V}(G)$, taken in topological order, **do**
[2] **for** each pair of identically labeled edges e_1, e_2 **do**
Perform edge merging and create new vertex
[3] Create a new vertex v having
$t(v) := \min(t(\beta(e_1)), t(\beta(e_2)))$,
inserting v into the topological order
Copy all edges incident from $\beta(e_1)$ *to* v
[4] **for** each edge $e = (\beta(e_1), w, s, y)$ **do**
[5] Create a new edge $e' := (v, w, s, y)$
Copy all edges incident from $\beta(e_2)$ *to* v
[6] **for** each edge $e = (\beta(e_2), w, s, y)$ **do**
[7] Create a new edge $e' := (v, w, s, y)$
Delete e_1, e_2
end

Algorithm 4. Reducing a graph to unique label sequences

The next step in finding a relevant measure of graph size rests on the observation that many different paths through a word graph produce exactly the same word sequence. Consequently, a further restriction would be to count only the number of distinct paths. In order to achieve this, the graph undergoes a transformation which changes the topology of the graph. However, this is not overly critical, since two identically labeled yet different paths may only differ in two aspects:

- The set of vertices which are visited by the paths. This information is usually irrelevant for a parser, since the exact mapping from words into the time space is normally of little importance. Yet, one has to notice that the integration of other knowledge sources, which may deliver information with tight time bounds — e.g. prosodic information about word or phrase boundaries or accents or information provided in different modalities, like deictic gestures, lip movements etc. — requires a strict invariance w.r.t. time.
- The acoustic scores which are annotated to word hypotheses. In the ideal case, an algorithm reducing a word graph to unique label sequences should preserve the relative scores of all paths through the graph. If that is not possible, as in our case, the score of a path should at least not get worse.

Algorithm 4 describes the process for reducing a word graph to unique label sequences. The algorithm ensures that no vertex is left by two edges having the same label. This local condition guarantees that, from a global perspective, there may not be two different paths describing the same sequence of word hypotheses. The method described here is similar to the algorithm (Hopcroft and Ullman, 1979) for converting a non-deterministic finite state automaton into a deterministic one (also cf. Seligman, Boitet and Hamrouni, 1998).

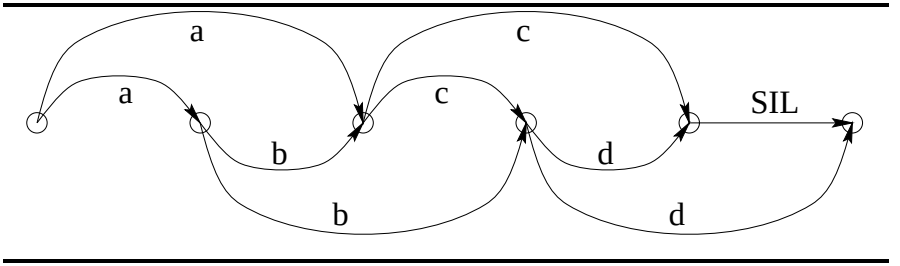


Figure 2.5. A difficult graph w.r.t the reduction to unique label sequences

The complexity of this algorithm is exponential in the worst case. Figure 2.5 shows a graph which clarifies this point. Each vertex, except the last two vertices, is the starting point of two identically labeled edges, one of which extends to the next vertex, and the other to the second next vertex. For each of these pairs, the algorithm has to create a new vertex, which has twice as many outgoing edges as the original vertex. If we, for the more general case, assume that each vertex has d pairwise identically labeled edges leaving it, the graph contains $2\,|\mathcal{V}| - 1$ edges.

The processing of the first vertex requires the construction of $\frac{d}{2}$ new vertices, each of which is equipped with $2\,d$ outgoing edges. If one of these new vertices is considered, again $\frac{d}{2}$ new vertices arise, this time with $4\,d$ outgoing edges each. This cycle can be repeated $\frac{|\mathcal{V}|}{2} - 1$ times, since we advance two vertices with every step. Throughout the course of considering vertices, we construct

$$\sum_{i=1}^{\frac{|\mathcal{V}|}{2}-1} 2^i\,d \tag{2.18}$$

new edges, which all have to be processed by the algorithm. This results in at least

$$d(2^{\frac{|\mathcal{V}|}{2}} - 2) \tag{2.19}$$

operations, the time complexity of the algorithm is thus $\Theta(\sqrt{2^{|\mathcal{V}|}})$.

The algorithm proposed here has not been applied to real word graphs, the complexity being too high. A modified version, which we will present in the following, belongs to the same complexity class, but some optimizations we carried out lead to acceptable runtimes. That way, we could carry out evaluations for word graphs.

Merging vertices has the most positive effect on runtime in this case. The merging of vertices described in Algorithm 5 copies edges from one vertex to another and carries out a redundancy test while doing so. Duplicate edges are not copied, only the acoustic score is modified if necessary. In order to keep the integrity of the word graph, only vertices not mutually reachable may be merged.

Merging of vertices may take place under different circumstances:

```
procedure Merge( v, w)
        Merge two vertices v and w
        First, copy w's ingoing edges
[1]     for each edge e = (x, w, s, l), do
[2]         if ∃e' = (x, v, s', l), then
[3]             s' ← min(s, s')
[4]         else
[5]             Create a new edge e'' = (x, v, s, l)

        Now, copy w's outgoing edges
[6]     for each edge e = (w, x, s, l), do
[7]         if ∃e' = (v, x, s', l), then
[8]             s' ← min(s, s')
[9]         else
[10]            Create a new edge e'' = (v, x, s, l)
end
```

Algorithm 5. Merging of two vertices

- Two vertices are merged if they have identical sets of edges incident to them and incident from them. However, this may lead to a slight increase in the number of paths since new partial paths of length two may be introduced by the merge operation.
- Two vertices are merged if they have identical sets of vertices incident from them.
- Two vertices are merged if they belong to the same point in time. This situation may arise as new vertices are created during the execution of the algorithm.

The merging of vertices is carried out whenever a new vertex is considered. In addition, merging may be executed if all the edges leaving a vertex have been processed. The vertices created during this part of the algorithm have identical sets of vertices incident to them (namely, the vertex just processed) and thus are potential candidates for merging.

By applying such modifications to the original algorithm, the runtime could be reduced to acceptable ranges; however, now scores of paths through the graph may be changed (the scores only get better), and even new paths may be introduced. Figure 2.6 shows a runtime overview for reducing graphs from the 1996 Verbmobil acoustic evaluation (Reinecke, 1996).[13]

So far, we have looked at the number of paths in a graph in order to approach a more realistic measure for word graph size than provided by the density. However, one has to observe that no linguistic system (e.g., a parser) would enumerate

[13] These are 305 utterances without spelling units from dialogs taken from Verbmobil CD 14.

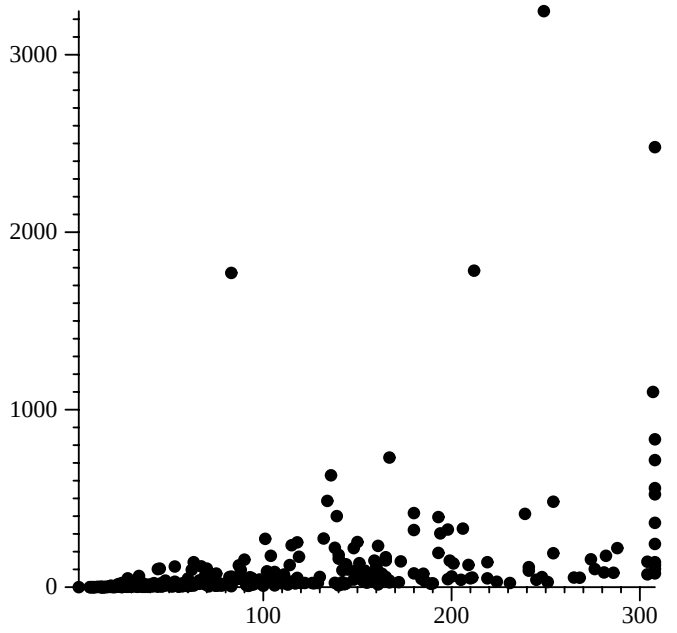


Figure 2.6. Runtime for reducing graphs to unique label sequences (x-Axis: log(Number of paths), y-Axis: Runtime in seconds)

all paths. Consequently, the last extension which we will pursue here targets the behavior of a reasonable, complex parser.

The fictitious parser, which we will view as the consumer of word graphs produced by a recognizer, shall use a unification-based feature structure formalism. This property prevents a cubic time complexity of the parser, as we are able to construct different sets of features for every pair of edges combined by a rule of the grammar (e.g. by concatenating the associated word hypotheses) (cf. Weber (1995) regarding the complexity of context-free word graph parsing, which is $O(|\mathcal{V}|^3)$). We restrict ourselves to grammars having a context-free backbone and feature structure annotations to rule elements, and additionally demand that grammars have Chomsky normal form, i.e. each rule has at most two nonterminals on the right side.[14] Other than that, we do not restrict the behavior of the parser, e.g. w.r.t its search methods.

begin

Initialization

[1] totalderiv $\longleftarrow 0$

Handle each vertex

[2] **for** each vertex $v \in \mathcal{V}(G)$, taken in topological order, **do**

Adjust the number of rule applications for this vertex and the total number of derivations so far.

[3] $\text{deriv}_v[1] \longleftarrow \#_{in}(v)$

[4] **for** all $i \in \{2, \ldots, |\mathcal{V}|\}$ do

[5] **for** each edge $e = (w, v, x, y)$ ending in v **do**

[6] $\text{deriv}_v[i] \longleftarrow \text{deriv}_v[i] + \text{deriv}_w[i-1]$

[7] totalderiv $\longleftarrow$ totalderiv $+ \text{deriv}_v[i]$

totalderiv holds the number of derivations

[8] **return** totalderiv

end

Algorithm 6. Calculation of the number of derivation steps of a fictitious parser for a word graph

As a measure of how much effort a parser has to invest in order to consume a graph, we take the number of elementary processing steps used for the complete analysis of the graph. If we look at the number of operations $d^{(p)}$ needed for the analysis of an utterance hypothesis p, then the following holds:

$$d^{(p)} = \frac{n^3 - n}{6} \tag{2.20}$$

where n is the length of the hypothesis in words. This number corresponds to the number of steps needed for filling the derivation matrix in the CKY-algorithm (cf. Mayer, 1986, p. 107).

If we assume that all paths within a graph are independent from each other, then the number of processing steps needed to analyze all paths in the graph is

$$d^{(G)} = \sum_{p \in G} d^{(p)} \tag{2.21}$$

This result suggests a linear dependency between the number of paths and the number of operations needed to process it. However, a fundamental property of

[14] As a grammar may encode ambiguity, the combination of two edges may render more than one result. We abstract away from this fact, as well as from the fact that due to restrictions formulated in the grammar not every combination of two edges may be valid.

begin

Compute minimum and maximum prefix scores

[1] **for** each vertex $v \in \mathcal{V}(G)$, taken in topological order, **do**

[2] $s_{v,min} \longleftarrow \infty$

[3] $s_{v,max} \longleftarrow 0$

[4] **for** each edge $e = (w, v, s_e, y)$ ending in v **do**

[5] $s_{v,min} \longleftarrow \min(s_{w,min} + s_e, s_{v,min})$

[6] $s_{v,max} \longleftarrow \max(s_{w,max} + s_e, s_{v,max})$

Compute rank of path with score s_{ref} recursively, starting at the final vertex

[7] $r \longleftarrow$ **ComputeRank** (v_f, s_{ref})

r is the rank of a path with score s_{ref} through the graph

[8] **return** r

end

Algorithm 7. Computing the rank of a path (part 1)

word graphs is the sharing of sub-paths, thus (2.21) is only an upper bound. In order to account for shared subgraphs, Algorithm 6 has to be applied.

Aside from the assumptions we have made about the properties of the grammar (lack of ambiguity and no restrictions about the applicability of rules), we further assume that the parser is free to derive an unlimited number of partial analyses at a vertex. A "reasonable" parser for spoken language would not do that, instead it would apply restriction mechanisms to keep the number of active hypotheses within a tractable range (*pruning*).

The complexity of Algorithm 6 is $O(|\mathcal{E}||\mathcal{V}|)$. The outer loop (line [2]) and the innermost loop (line [5]) guarantee that all edges are considered and impose an order on their processing. The loop in line [4] increases the number of derivations up to the current vertex. All shorter analyses constructed so far are used. These sequences have a maximum length of $|\mathcal{V}|$, thus leading to the complexity figure we gave.

2.4 Evaluation of Word Graphs: Quality Measures

In the preceding section, we discussed several different size measures for word graphs; we presented an important modification which influences the size of a word graph to a great degree. We only implicitly mentioned quality measures, though. In this section, we will introduce different quality measures, all of which are based on the word accuracy in that they refer to the "best chain" through the graph and

assess some property of that chain. Restricting oneself to word accuracy alone can be dangerous, as this neglects the shape of the underlying word graph (just as all edge-based size measures do). In principle, other integrated measures are possible, which take into account the topology of the word graph. The quality of a word graph with respect to the reference utterance could be computed as follows:

- The difference in acoustic scores between the chain with the best acoustic score and the "correct" chain[15].
- The probability of the correct chain w.r.t. the input graph.
- The number of derivation steps an ideal parser would need in order to find the correct chain.
- The rank of the correct chain if only acoustic scores are taken into account.

The motivation to use the rank of the correct chain in the list of all paths through the graph ordered by overall acoustic score is again driven by the assumption that a subsequent parser considers all paths in the worst case, until the correct one is found. Of course, this assumption does not take into account that a parser could well find a different path with a higher acoustic score, for which it could assign a syntactic analysis. The method to compute the rank of a path within a word graph is described in Algorithm 7. More precisely, the algorithm computes the rank of a chain with a given overall acoustic score w.r.t. the scores of all other paths through the graph. Thus, the input parameter for the algorithm consists of a word graph and a reference score s_{ref}, the output is the rank of a path with that score.

A naive search enumerates all paths in the order of decreasing acoustic score until one is found that carries the score which is sought. The algorithm described here is founded on eliminating edges at the end of a path and to compare minimal and maximal scores on the way. Thereby, in some situations, numerous paths can be handled in one single step; this should lead to a more efficient solution for many cases compared to a simple best-first search based on acoustic scores.

However, in the worst case the algorithm still shows a complexity which is exponential in the number of vertices in a graph. The reason for this property is that — given a disadvantageous distribution of scores — the complete graph still has to be considered. Let us assume a graph with $|\mathcal{V}|$ vertices and $n(|\mathcal{V}|-1)$ edges for any odd n. Figure 2.7 shows such a graph with four vertices and $n = 5$.

The edges are distributed evenly between the vertices of the graph, with n edges between two neighboring vertices. Let the vertices have numbers 1 through $|\mathcal{V}|$ according to their topological ordering. Let the weights (scores) of the edges between the vertex i and the vertex $i+1$ be as follows:

- The edge belonging to the reference path bears the average weight, for instance, 0.
- Half of the remaining edges bear the weight $\frac{1}{2^i}$.
- All remaining edges bear the weight $-\frac{1}{2^i}$.

[15] By "correct" chain we mean the path through the graph which gives the highest word accuracy w.r.t the reference transliteration.

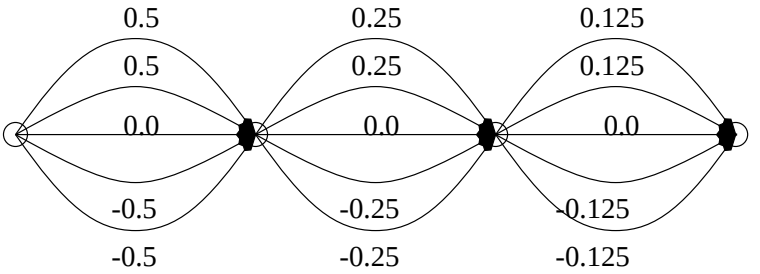


Figure 2.7. A complex graph for rank computation

function ComputeRank (v, s_{ref} **)**

If the reference score is lower than or equal to the minimum score, all paths to the left are worse and need not be considered

[1] **if** $s_{ref} \leq s_{v,min}$

[2] **return** 0

If the reference score is greater than or equal to the maximum score, all paths to the left are better and need not be considered

[3] **if** $s_{ref} \geq s_{v,max}$

[4] **return** $p^{(v)}$

Now comes the hard part. Recursively consider all vertices incident to v

[5] $r_{tmp} \longleftarrow 0$

[5] **for** each edge $e = (w, v, s_e, y)$ ending in v **do**

[6] $r_{tmp} \longleftarrow r_{tmp}$+**ComputeRank (**$w, s_{ref} - s_e$**)**

[7] **return** r_{tmp}

end

Algorithm 8. Computing the rank of a path (part 2)

The lines [1] to [6] of Algorithm 7 determine the lower and upper bounds of scores of possible paths from the root of the graph to the current vertex. Then, the function **ComputeRank()**, which is shown as Algorithm 8, forms the core of the computation of the rank of a chain. If the score sought is smaller than the minimal score or greater than the maximal score up to the current vertex, then all vertices to the left of the current vertex do not need to be considered. The solution (how many paths to the left are better or worse than the reference) can be determined from $s_{v,max}$ or $s_{v,min}$. If, however, the score falls in the interval $]s_{v,min}, s_{v,max}[$, then a recursive operation is needed.

Applied to the example graph, the function is called first at the end vertex with a reference score of 0. As this score is neither smaller than the lowest nor greater than the biggest score, n subsequent calls to **ComputeRank()** result from this. As no partial path from any internal vertex to the end vertex of the graph can have a cumulative score with an absolute value greater than any edge ending at the start vertex of the partial path, it follows that at each vertex n new calls to **ComputeRank()** are generated.

It follows that in total

$$\prod_{i=1}^{|\mathcal{V}|-1} n \quad = \quad n^{|\mathcal{V}|-1} \tag{2.22}$$

calls are executed, thus the complexity in the worst case is $\Theta(n^{|\mathcal{V}|})$.

All quality measures we mentioned are derived from word accuracy. Here, we only described the last possibility, but for evaluation of word graphs we will exclusively use word accuracy from now on. The reasons for this lie, on one hand, in the desired comparability to best-chain recognizers, and on the other hand, the inherent problems with graph evaluation.

All alternative evaluation measures introduced in this section try to give some significance to how difficult it is to actually find the correct result. However, this measure alone is not sufficient, as it leads to the same value for all best-chain recognizers. Thus, an evaluation has to deliver at least a pair of results: A quality measure (like the word accuracy) and an effort measure (like the rank of the correct chain).

But even this does not necessarily give a precise picture of how much effort a system will actually spend for processing a word graph. Depending on the kind of modules or the strictness of grammars applied by those modules, a system will not explore the complete search space up to the chain referenced by the effort measure. Modern systems are often constructed with extremely general knowledge sources which assign analyses to a broad range of input utterances. In the extreme case, this may lead to a syntactic analysis for the chain with the best overall acoustic score, even if the word accuracy for that chain is much lower than theoretically possible.

On the other hand, a word accuracy rate is a flat measure. It gives no details about which words were actually recognized and what kind of errors occurred. In the worst case, this may lead to a recognition result that misses all the important content words and where only irrelevant parts of the utterance are recognized correctly.

To obtain an integrated measure of word graph quality, which correctly estimates "success chances" for linguistic analysis and gives a good figure of how much effort is needed to obtain the analysis, is nearly impossible in our view. The consequence we draw from this is to continue to use the word accuracy as a quality measure and to only favor a change in size measures for word graphs.

2.5 Further Operations on Word Graphs

Edges annotated with silence (!SIL) are worth separate consideration. The reason is that it is almost always assumed that they carry no meaning and can be ignored within modules for linguistic processing. For instance, the insertion of silence edges in word graphs is not regarded as an error during evaluation.[16] Following this insight, one possible way to augment the results of a speech recognizer would be to remove as many silence edges from a word graph as possible. This does not change the word accuracy, but it reduces the density of the graph (by removing edges) and thus increases the accuracy of the recognizer in relation to the graph size. We will present three different methods for dealing with silence. Afterwards, we present an algorithm which increases the accuracy of a recognizer by employing the current evaluation metrics in an unorthodox way.

2.5.1 Removing Isolated Silence

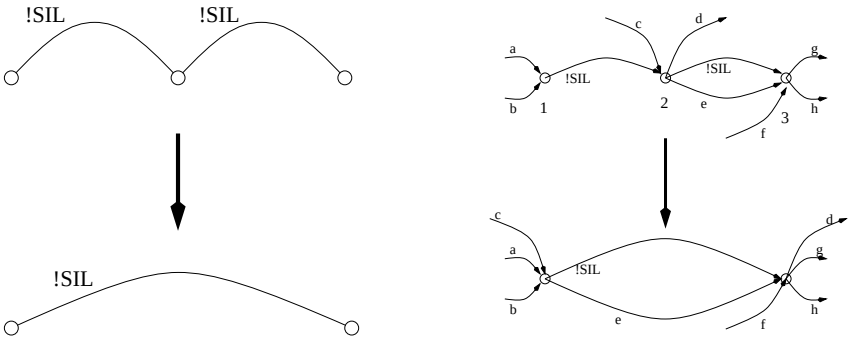
The first algorithm we present here is the removal of single silence edges connecting two vertices of a word graph. It assumes that the silence edge is the only edge connecting the vertices. Usually, this happens only at the start and at the end of a word graph, which by definition carry silence edges. The number of cases where this algorithm is applicable rises, however, if the word graph is modified by other means described earlier.

The main idea of Algorithm 9 is to merge two vertices if they are exclusively connected by a silence edge. Naturally, the affected vertices must not be reachable from one another by other edges, in order to keep the integrity of the graph. One of the versions of this algorithm introduced an additional constraint which kept the vertices in chronological order. The merging of vertices would not have been carried out if this would have resulted in edges that point backwards in time. However, this restriction is not essential for Algorithm 9, and was left out. In any case, the copied edges have to be rescored in order to account for the acoustic scores of the removed silence edge; the overall scores for paths should not change.

[16] From a linguistic point of view, pauses can be highly relevant. For example, they may be used to trigger the application of specific syntactic rules in grammars that describe turns with several partial utterances (Kasper and Krieger, 1996). In those cases, a specialized recognizer for prosodic events should be used (Strom and Widera, 1996).

begin
[1] **for** each vertex $v \in \mathcal{V}$, taken in topological order, **do**
Search for silences leaving v
[2] **for** each edge $e = (v, w, s, \mathsf{!SIL})$ **do**
[3] **if** $v \overset{*}{\not\to} w$, disregarding e, **then**
Adjust scores
[4] **for** each edge $e' = (v, w', s', l'), e \neq e$, **do**
[5] $s' \longleftarrow s' + s$
Merge vertices v and w
[6] **Merge(**v, w**)**
[7] Delete w
end

Algorithm 9. Removing isolated silence edges



a: Exclusively silence

b: Silence and other edges

Figure 2.8. Merging of silence edges

2.5.2 Removing Consecutive Silence

The only type of word hypothesis whose repetition is meaningless is silence. If two silence edges appear in a row, they can be merged. The simplest situation is the one shown in Figure 2.8 a). Both edges can be replaced by a single edge and the intervening vertex can be removed. However, this case is relatively rare and surfaces only when word hypotheses have been removed from the intervening vertex.

The more interesting situation is shown in Figure 2.8 b). Edges that begin or end at the intervening vertex have to be treated specially in order to prevent major modifications within the graph. We restrict ourselves to cases that do not have edges lying in the time interval spanned by the outer vertices, thereby preventing cycles. There are several possibilities for treating the edges adjacent to the intervening ver-

begin
[1] **for** each vertex $v \in \mathcal{V}$, taken in topological order, **do**
Search for two silences
[2] **if** $\exists e_1 = (v_1, v, s, !\mathsf{SIL}), e_2 = (v, v_2, s', !\mathsf{SIL}) \in \mathcal{E}$, **then**

Check applicability
[3] **for** each edge $e = (w, v, x, y) \in \mathcal{E}$, **do**
[4] **if** e falls into the range $[t(v_1), t(v_2)]$, **then**
[5] **continue** with loop in line **[2]**

Move edges incident to v along
[6] **for** each edge $e = (v', v, x, y) \in \mathcal{E}, e \neq e_1$, **do**
[7] **if** $v' = v_1$, **then**
[8] Create new edge $e' = (v_1, v_2, x + s', y)$
[9] **else**
[10] Create new edge $e' = (v', v_1, x - s, y)$

Move edges incident from v along
[11] **for** each edge $e = (v, v', x, y) \in \mathcal{E}, e \neq e_1$, **do**
[12] **if** $v' = v_2$, **then**
[13] Create new edge $e' = (v_1, v_2, x + s, y)$
[14] **else**
[15] Create new edge $e' = (v_2, v', x - s', y)$

[16] Delete v and all incident edges
end

Algorithm 10. Removing consecutive silence

tex. The state of affairs in Figure 2.8 b) is such that edges incident to vertex 2 are moved into vertex 1, while edges incident from vertex 2 are moved into vertex 3. By doing that, new partial paths may be introduced into the graph (here: the edge sequence $c - e - d$). If, on the other hand, d is bound to vertex 1 and c to vertex 3 and paths may vanish from the graph.[17]

By moving edges to the border vertices, the number of edges incident to or from them increases. This, in turn, increases the probability of reducing the number of paths simply by removing identical word hypotheses. This usually leads to good results. The mechanism to remove consecutive silence is given in Algorithm 10.

[17] We do not discuss the two remaining possibilities here.

```
begin
[1]     for each vertex v ∈ V, taken in topological order, do
[2]         for each edge e = (v, w, s, !SIL), do
                Copy edges from the endvertex
[3]             for each edge e' = (w, x, t, l), do
[4]                 if ∃e'' = (v, x, t', l), then
[5]                     t' ←— min(t + s, t')
[6]                 else
[7]                     Create a new edge e'' = (v, x, t + s, l)
[8]             Delete e
end
```

Algorithm 11. Removing all silence edges

2.5.3 Removing All Silence Edges

The most radical treatment of silence edges is to remove them altogether from a word graph. Algorithm 11 describes the method. All edges which start at the end vertex of a silence edge are copied into the start vertex of the silence edge, and their scores are modified accordingly. However, this must not happen at the end of the word graph, in order to prevent multiple end vertices for a graph.

The copying of edges clearly increases the number of edges in a graph. However, the number of paths can be reduced by removing identical copies of edges. Moreover, the application of other methods, like the reduction to identical label sequences, now has a high probability of decreasing the number of paths.

2.5.4 Merging Mutually Unreachable Vertices

The choice of evaluation criteria for word recognizers sometimes leads to interesting methods for their augmentation. In the course of the recognizer evaluation for the Verbmobil project in 1996 the measure for the size of word graphs was the density, i.e. the number of edges per word in the reference transliteration. Within some classes defined as density intervals, the word accuracy was selected as a quality measure.

Consequently, the goal of the graph modification efforts was to combine a high number of paths through the graph with a low number of edges. One possible approach is to merge vertices whenever possible. The merging decreases the number of edges (e.g., if edges occur that carry the same label), but the main effect is gained by increasing the number of paths. For statistical reasons alone, the word accuracy increases. Besides the algorithms we mentioned before, which remove vertices by considering silence edges, the most radical approach would be to merge vertices which are mutually not reachable. The topology of the graph is not disturbed (in

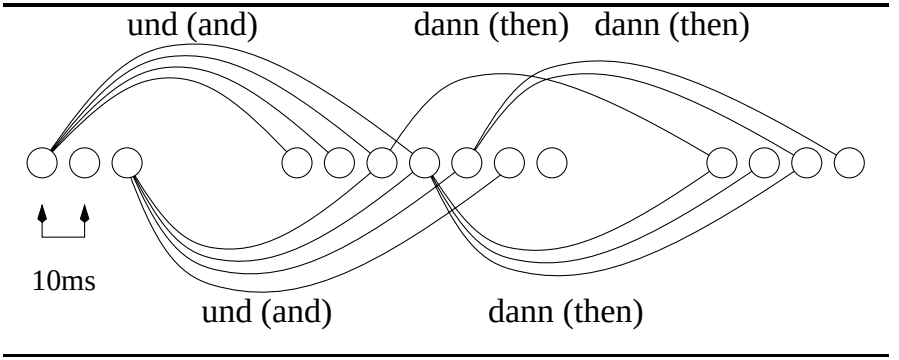


Figure 2.9. Two families of edges in a word graph

particular, no cycles are introduced), the number of edges decreases and the number of paths increases drastically. During the evaluation we mentioned, using this method the word accuracy could be increased by ca. 1.5%. Of course, graphs that undergo such a modification would be essentially useless for any kind of linguistic analysis. Vertices belonging to completely different intervals of the input signals could be merged; the word sequences are rendered meaningless.

2.6 Hypergraphs

The biggest problem in processing word graphs within natural language processing systems is, of course, their size (regardless of the actual choice of a size measure, word graphs — in particular incremental word graphs — *are* extremely big and accordingly difficult to process). One of the main reasons for this is the occurrence of several almost identical word hypotheses. By "almost identical", we mean that two edges bear the same word hypotheses, but with slightly different start and end vertices. Figure 2.9 shows a small fictitious part of a word graph, which contains many hypotheses representing the words und and dann. The start and end vertices differ only by a few hundredths of a second.

The existence of such *families* of edges (Weber, 1995) stems from at least two distinct reasons:

- Word recognizers using hidden markov models — like the Hamburg word recognizer (Huebener, Jost and Heine, 1996) used to create the graphs investigated in this work — attempt to start and end word models for each time frame. As the chronological resolution is quite high (usually, frames with a length of 10ms are used), this results in multiple identical hypotheses in a number of consecutive vertices.

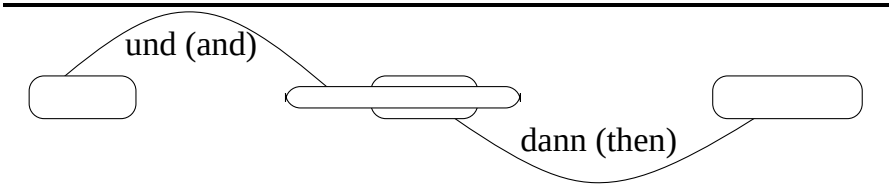


Figure 2.10. An interval graph

- Spoken language tends to blur word boundaries. This effect is partly responsible for the drastic decrease in word accuracy that can be experienced when moving from controlled speech with accentuated pauses to continuous speech. This is particularly true for spontaneous speech. Figure 2.9 demonstrates this uncertainty by having several connecting vertices between the hypotheses und and dann.

Thus, a speech recognizer usually issues bundles of word hypotheses with connections in several vertices. Both properties pose a great burden on linguistic processing. A high number of initial word hypotheses results in a high number of lexical access steps and basic operations, e.g. the proposal of syntactic categories during a bottom-up parsing. Many connecting vertices lead to a high number of complex operations, for instance unifications combining the linguistic descriptions of edges.

There are a number of possibilities to tackle this problem:

1. An obvious solution would be to reduce the resolution of the recognizer within linguistic processing (cf. Weber, 1992). Whenever a module for linguistic processing receives a word hypothesis, the start and end times are mapped into a coarser resolution; a redundancy test is used to avoid the introduction of multiple identical hypotheses. This approach is simple, yet it leads to the introduction of spurious paths by artificially creating connecting vertices for edges that would otherwise not have been connected. Moreover, the correct choice of a resolution is difficult, as the length of the necessary intervals at word onsets and offsets are different for different words.
2. A direct and more consistent choice is to use interval graphs as a representation means for word hypotheses. Edges no longer connect two distinct vertices, but we could use intervals to represent the start and end times. Figure 2.10 shows such a graph. The main problem with this kind of representation lies in the complexity of edge access. However, some of the properties shown later can be proven easier and without loss of generality using interval graphs.
3. The method which seems to us to be the most appropriate and that we consequently use throughout this work uses hypergraphs to represent word graphs and interpretation graphs.[18] The treatment is founded on the maxim that opera-

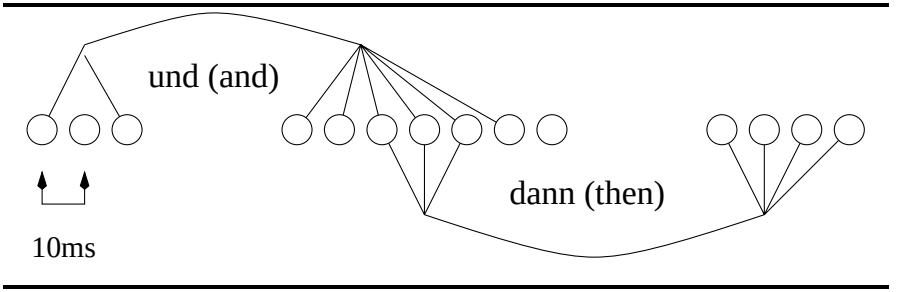


Figure 2.11. Two families of word hypotheses as hyperedges

tions should be carried out once. If almost identical edges are encountered, this should be simply annotated to the edge, which then has several start and end vertices. Weber (1995) introduces the notion of a *family of edges* for a set of edges bearing the same label and having the same start vertex, but different end vertices. In the framework presented here, we extended his findings to edges with different start vertices as well, which led to an additional edge reduction of 6% (Amtrup and Weber, 1998). Figure 2.11 shows the familiar graph as a hypergraph. In order to process such graphs, we adapt the method of Weber (1995) of modifying the acoustic scores of edges.

2.6.1 Formal Definition of Hypergraphs

Hypergraphs (cf. Gondran and Minoux, 1984, p. 30ff) are constructed from undirected graphs by viewing sets of vertices as edges. In the following, we always use directed hypergraphs (Gondran and Minoux, 1984, p. 33), and call them hypergraphs for brevity. First, we are going to describe hypergraphs as an extension of word graphs; however, the hypergraph property can be easily extended to capture incremental word graphs as well.

Definition 2.6.1. *Hypergraph*
A hypergraph is a quadruple $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{W})$ *with*

- *a set* $\mathcal{V}$ *of vertices as defined in Definition 2.2.2,*
- *a set* $\mathcal{L}$ *of edge labels as defined in Definition 2.2.2,*
- *a set* $\mathcal{W}$ *of edge weights as defined in Definition 2.2.2,*
- *and a set* $\mathcal{E}$ *of hyperedges with*

$$\mathcal{E} \subseteq \mathcal{V}^* \backslash \emptyset \times \mathcal{V}^* \backslash \emptyset \times \mathcal{W} \times \mathcal{L} \tag{2.23}$$

[18] The formalization and algorithmic treatment of hypergraphs were developed in cooperation with Volker Weber, cf. Weber (Forthcoming).

Some of the notations and functions we defined for word graphs have to be adapted in order to be used for hypergraphs. For the following, let $e = (V, V', w, l)$ be a hyperedge.

- *The access functions for start and end vertices of hyperedges return sets of vertices:*

$$\alpha : \mathcal{E} \longrightarrow \mathcal{V}^*, \alpha(e) := V \tag{2.24}$$
$$\beta : \mathcal{E} \longrightarrow \mathcal{V}^*, \beta(e) := V' \tag{2.25}$$

- *Two hyperedges e and e' are adjacent if they share at least one vertex, i.e. iff:*

$$\beta(e) \cap \alpha(e') \neq \emptyset \tag{2.26}$$

- *The reachability relation is now*

$$\forall v, w \in \mathcal{V} : v \rightarrow w \Longleftrightarrow \exists e \in \mathcal{E} : v \in \alpha(e) \wedge w \in \beta(e) \tag{2.27}$$

Additionally, we define access functions for the extreme start and end vertices, as well as for the intervals which are covered by start and end vertices:[19]

$$\alpha_< : \quad \mathcal{E} \longrightarrow \mathcal{V}, \quad \alpha_<(e) := \mathit{arg\,min}\{t(v) | v \in V\} \tag{2.28}$$
$$\alpha_> : \quad \mathcal{E} \longrightarrow \mathcal{V}, \quad \alpha_>(e) := \mathit{arg\,max}\{t(v) | v \in V\} \tag{2.29}$$
$$\beta_< : \quad \mathcal{E} \longrightarrow \mathcal{V}, \quad \beta_<(e) := \mathit{arg\,min}\{t(v) | v \in V'\} \tag{2.30}$$
$$\beta_> : \quad \mathcal{E} \longrightarrow \mathcal{V}, \quad \beta_>(e) := \mathit{arg\,max}\{t(v) | v \in V'\} \tag{2.31}$$
$$\alpha_{[]}(e) := [t(\alpha_<(e)), t(\alpha_>(e))] \tag{2.32}$$
$$\beta_{[]}(e) := [t(\beta_<(e)), t(\beta_>(e))] \tag{2.33}$$

The definitions given above assume that each vertex is associated with a certain point in time. Although it is in principle not necessary to introduce a total ordering on the set of vertices, this is implicitly done by the definition of hypergraphs. However, since speech is by nature uttered linearly in time, we do not feel that this is too strong of a restriction.

It is helpful to define the hyperedge belonging to a word edge $e = (v, v', w, l)$. We write $\mathcal{H}e$ and set $\mathcal{H}e = (\{v\}, \{v'\}, w, l)$.

In contrast to interval graphs, we do not demand that the set of start and end vertices be compact, i.e. neither do we demand

$$\forall e \in \mathcal{E}, v \in \mathcal{V} \quad : \quad \alpha_<(e) \leq t(v) \leq \alpha_>(e) \Longrightarrow v \in \alpha(e) \quad \text{, nor} \tag{2.34}$$
$$\forall e \in \mathcal{E}, v \in \mathcal{V} \quad : \quad \beta_<(e) \leq t(v) \leq \beta_>(e) \Longrightarrow v \in \beta(e) \tag{2.35}$$

If the individual vertices from $\alpha(e)$ or $\beta(e)$ are irrelevant in a description, we implicitly use interval graphs for the sake of simplicity of an argument.

Hypergraphs should be acyclic just like word graphs in order to be easy to process. Thus, at least the following should hold:

[19] The function arg min returns the argument that belongs to the minimum value, not the value itself. For instance, $\arg\min\{x^2 | x \in \{2, 3, 5\}\}$ returns 2, and not 4.

$$\forall v \xrightarrow{*} w : v \neq w \tag{2.36}$$

Moreover, we demand that

$$\forall e : t(\alpha_>(e)) < t(\beta_<(e)) \tag{2.37}$$

The second requirement is more strict than 2.36, but can be motivated from the linear ordering of speech in time, and it prevents a hyperedge from being directed backwards in time.

2.6.2 Merging of Hyperedges

Adding a simple word hypothesis to a hypergraph is a special case of merging two hyperedges, as we can view each word hypothesis as a hyperedge with exactly one start and one end vertex. We will discuss the general case first. In order to merge two hyperedges without loss of linguistic information, three conditions must hold:

- The lexical entries belonging to the hyperedges must be identical,
- the edge weights (e.g. acoustic scores) must be combined in an appropriate manner, and
- the sets of start and end vertices must be compatible and must be combined without the introduction of cycles.

Definition 2.6.2. *Merging of hyperedges*
Let $e_1, e_2 \in \mathcal{E}$ be two hyperedges of a hypergraph G with $e_1 = (V_1, V_1', w_1, l_1)$ and $e_2 = (V_2, V_2', w_2, l_2)$. These hyperedges may be combined, iff

$$l(e_1) = l(e_2) \tag{2.38}$$

$$\min(t(\beta_<(e_1)), t(\beta_<(e_2))) > \max(t(\alpha_>(e_1)), t(\alpha_>(e_2))) \tag{2.39}$$

The combination results in a new hyperedge $e_3 = (V_3, V_3', w_3, l_3)$ with the new components

$$l_3 = l_1 \quad (= l_2) \tag{2.40}$$

$$w_3 = \textit{scorejoin}(e_1, e_2) \qquad \textit{(see below)} \tag{2.41}$$

$$V_3 = V_1 \cup V_2 \tag{2.42}$$

$$V_3' = V_1' \cup V_2' \tag{2.43}$$

e_1 and e_2 are removed from G, e_3 is inserted.

The two conditions mentioned are sufficient to guarantee a consistent merging of hyperedges. Condition (2.38) avoids the merging of hyperedges belonging to different word hypotheses, and condition (2.39) states which hyperedges may be combined and prevents cycles. A short analysis of the cases that may arise shows this. Without loss of generality, we assume that $t(\beta_>(e_1)) \leq t(\beta_>(e_2))$, i.e. e_2 ends behind e_1.

- $\alpha_{[]}(e_1) \cap \beta_{[]}(e_2) \neq \emptyset \quad \vee \quad \alpha_{[]}(e_2) \cap \beta_{[]}(e_1) \neq \emptyset$
  The start vertices of one edge overlap with the end vertices of the other. A combination of both edges would result in an edge e_3 with $t(\alpha_>(e_3)) \geq t(\beta_<(e_3))$. As this would introduce cycles in the graph, the condition (2.39) prevents this situation.
- $\alpha_{[]}(e_1) \cap \beta_{[]}(e_2) = \emptyset \quad \wedge \quad \alpha_{[]}(e_2) \cap \beta_{[]}(e_1) = \emptyset$
  This is the inverse case.
  - $t(\alpha_<(e_2)) \geq t(\beta_>(e_1))$
    This is the case where all vertices of hyperedge e_1 occur before all vertices of hyperedge e_2. In other words, this represents the case where two individual independent word hypotheses with the same label occur in the word graph, for instance because the speaker uttered the same word twice. This case must also not result in an edge merge since $\beta_{[]}(e_1) \subseteq [t(\alpha_<(e_1)), t(\alpha_>(e_2))]$ in the merged edge. This merge is prohibited by condition (2.39) since all vertices of $\beta(e_1)$ have to be smaller than all vertices of $\alpha(e_2)$.
  - $t(\alpha_<(e_2)) < t(\beta_>(e_1))$
    This is the inverse case.
    - $t(\alpha_<(e_1)) \geq t(\beta_>(e_2))$
      This case does not arise due to the consistency of the edges and the additional assumption that $t(\beta_>(e_1)) \leq t(\beta_>(e_2))$, i.e. e_2 contains the last end vertex.
    - $t(\alpha_<(e_1)) < t(\beta_>(e_2))$
      This is the inverse case. The hyperedges are arranged such that $t(\alpha_>(e_1)) < t(\beta_<(e_2))$ and $t(\alpha_>(e_2)) < t(\beta_<(e_1))$ hold. Thus, $\forall t_\alpha \in \alpha_{[]}(e_1) \cup \alpha_{[]}(e_2), t_\beta \in \beta_{[]}(e_1) \cup \beta_{[]}(e_2) : t_\alpha < t_\beta$. This is exactly the condition stated in (2.39), both edges may be merged.

The question of how to combine the weights of two hyperedges while merging them can only be answered on an application-dependent basis. If we assume that we only need to consider word graphs and similar constructions, as is reasonable here, then we can define the function scorejoin() as follows. The result is to use the score that represents the best (smallest) value per frame. The score of an edge is normalized from the earliest start vertex to the latest end vertex.

$$\begin{aligned} \text{scorejoin}(e_1, e_2) := \quad & min\left(\frac{w_1}{t(\beta_>(e_1)) - t(\alpha_<(e_1))}, \frac{w_2}{t(\beta_>(e_2)) - t(\alpha_<(e_2))}\right) \cdot \\ & \cdot (t(\beta_>(e_n)) - t(\alpha_<(e_n))) \end{aligned} \tag{2.44}$$

As already mentioned, the insertion of word hypotheses into a hypergraph is a special case of the situation just described. The task is to merge a word hypothesis with a preexisting hyperedge. The relevant relative positions are pictured in Figure 2.12.

The already existing hyperedge is called e_g, e_1 to e_5 are candidates that could probably be merged into the hyperedge. e_1 and e_2 cannot be merged, since this would introduce cycles in the graph. The results of inserting e_3 to e_5 are shown in the figure. The mechanism of adding edges is described in Algorithm 12.

Each hyperedge is assigned a single weight. Starting from the acoustic scores of the original word hypotheses, the relative acoustic score per frame of 10ms seems

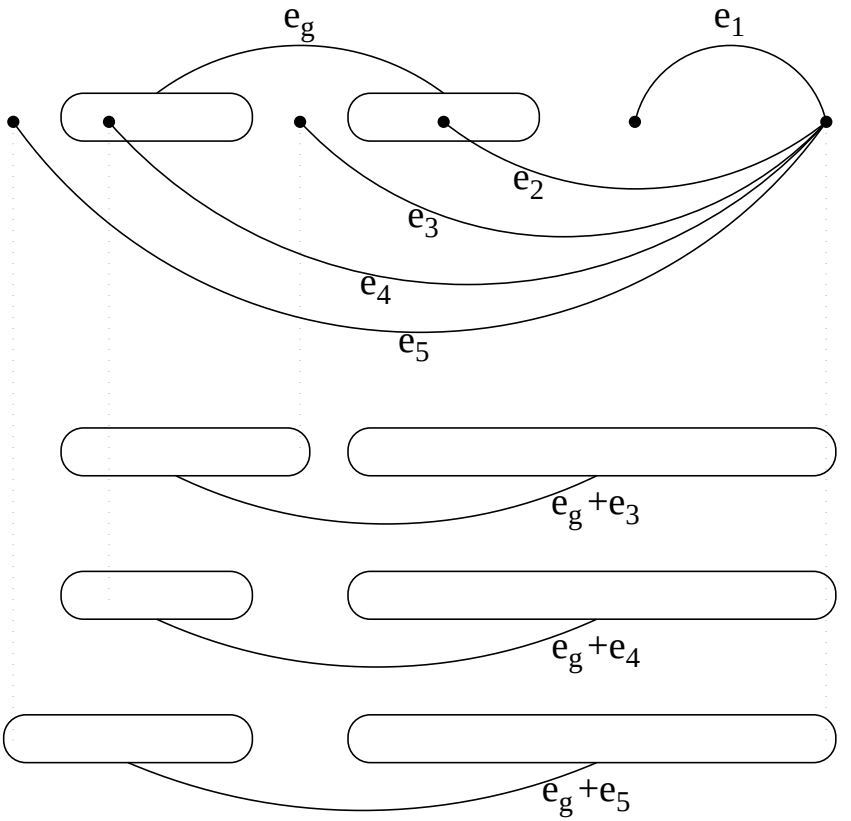


Figure 2.12. Adding a word hypothesis to a hyperedge

begin
[1] **if** $\exists e_k \in \mathcal{E}$ **with** $l(e_k) = l(e_n) \wedge t(\beta_<(e_k)) > t(\alpha(e_n))$ **then**
Modify edge e_k
[2] $e'_k := (\alpha(e_k) \cup \{\alpha(e_n)\}, \beta(e_k) \cup \{\beta(e_n)\}, \text{scorejoin}(w(e_k), w(\mathcal{H}e_n)), l(e_k))$
[3] **return** $G' := (\mathcal{V} \cup \{\alpha(e_n), \beta(e_n)\}, \mathcal{E} \backslash e_k \cup \{e'_k\}, \mathcal{W} \backslash w(e_k) \cup \{w(e'_k)\}, \mathcal{L})$
[4] **else**
Add edge $\mathcal{H}e_n$
[5] **return** $G' := (\mathcal{V} \cup \{\alpha(e_n), \beta(e_n)\}, \mathcal{E} \cup \{\mathcal{H}e_n\}, \mathcal{W} \cup \{w(e_n)\}, \mathcal{L} \cup \{l(e_n)\})$
end

Algorithm 12. Adding a word hypothesis e_n to a hypergraph $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{W})$

to be a suitable choice (cf. Weber, 1995). We use the minimal score per frame as the weight of the hyperedge.[20] By doing this, paths through the hypergraph can only be assigned better scores than the corresponding paths through the original word graph.

2.6.3 Combination of Hyperedges

The combination of hyperedges in a graph, e.g. during syntactic analysis, is completely analogous to the corresponding operations on word graphs. Two hyperedges e_1 and e_2 may be combined if they are adjacent, i.e. if the following holds:

$$\beta(e_1) \cap \alpha(e_2) \neq \emptyset \tag{2.45}$$

The symmetric case ($\beta(e_2) \cap \alpha(e_1) \neq \emptyset$) is treated accordingly. Usually, there have to be additional well-formedness criteria, which are imposed by the knowledge sources being applied (grammars, etc.). From both hyperedges, we construct a new hyperedge e_n, which inherits the start vertices of one edge and the end vertices of the other:

$$\alpha(e_n) := \alpha(e_1) \tag{2.46}$$

$$\beta(e_n) := \beta(e_2) \tag{2.47}$$

Due to the consistency of the originating edges and the non-empty intersection of start and end vertices, the new edge is consistent as well. We either use generic names as labels for new edges, or the labels are derived from the involved knowledge sources.

For the calculation of the weight of the new edge e_n — in our case primarily the acoustic score — we again use the smallest possible score value in analogy to the merging of two hyperedges. Since $\beta(e_2) \cap \alpha(e_1)$ may contain several vertices that correspond to different points in time, we have to search for the minimum of the combined scores of $w(e_1)$ and $w(e_2)$:

$$w(e_n) = \min_{\forall v \in \beta(e_2) \cap \alpha(e_1)} \frac{w(e_1) \cdot (t(v) - t(\alpha_<(e_1))) + w(e_2) \cdot (t(\beta_>(e_2)) - t(v))}{t(\beta_>(e_2)) - t(\alpha_<(e_1))}, \tag{2.48}$$

$w(e_n)$ denotes the acoustic score in case e_1 and e_2 occur one after another in the input. Due to inheritance mechanisms, the weight of e_n has to be modified if one of the scores of the originating edges is changed or if one of the vertex sets of the edges changes. The type of calculation we described here does not yet include additional evidence, for instance by probabilistic grammars, language models etc.; however, the adaptation of such cases can be done without any problem.

The method we described in this section for combining hyperedges seems reasonable, since the number of edges in a graph can be drastically reduced. However,

[20] As scores are represented by negative logarithms of densities, a smaller value reflects a higher probability.

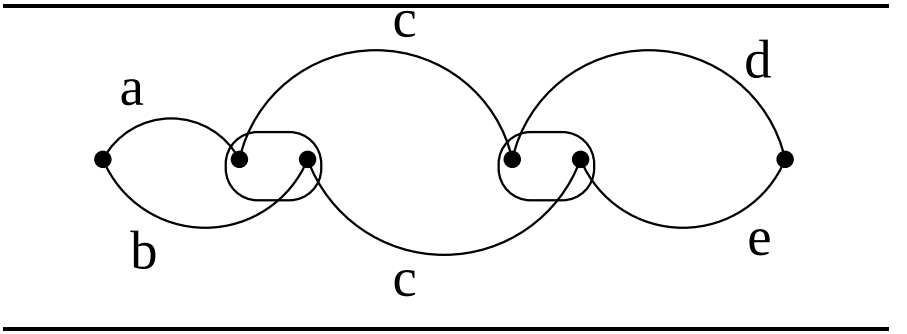


Figure 2.13. Creation of additional paths by using hypergraphs

by using hypergraphs the combination of edges may result in additional, spurious paths in a graph. Figure 2.13 shows that situation. If the graph shown here is treated as an ordinary word graph, it contains two label sequences, namely a-c-d and b-c-e. Inducing a hypergraph with these data creates a hyperedge with the label c, which has two start vertices and two end vertices. By doing so, we introduce two new paths, namely a-c-e and b-c-d. Thus, the transition to hypergraphs is not information preserving in a strong sense of the word. However, this does not play an important role in practical applications, since the situations in which new paths could be introduced are extremely rare. We have not experienced any problems during our experiments. By conversion to hypergraphs, only 0.2% additional analyses have been introduced (cf. Section 5.1).

2.7 Search in Graphs

The goal of processing an input during any kind of natural language processing is usually to find an interpretation which is optimal according to some success criteria chosen. In the application to the data structures used in this work, this means finding a path through a word graph whose weight sum is optimal. Usually there are algorithms for finding shortest paths (cf., e.g. Brandstädt, 1994). On the level of acoustic scores this corresponds to the commonly used representation by negative logarithms of densities.

The question at hand is the following: Which is the shortest path from the start vertex of a graph to its end vertex, and what is its weight? It has been shown that the computation does not become more difficult if the more general problem of the single source shortest path is solved. SSSP assigns a weight to each vertex, which is the weight of the shortest path from some special start vertex. The general case can be computed by the algorithm of Bellman-Ford (Cormen, Leiserson and

```
begin
        Initialize shortest path estimate and minimal weight
[1]     for each vertex v ∈ V(G) do
[2]         d[v] ⟵ ∞
[3]         π[v] ⟵ NIL
[4]     d[s] ⟵ 0

        Construct shortest paths
[5]     for each vertex u ∈ V(G), taken in topological order do
[6]         for each vertex v ∈ Adj(u) do
                Relax the edge
[7]             if d[v] > d[u] + w(u, v) then
[8]                 d[v] ⟵ d[u] + w(u, v)
[9]                 π[v] ⟵ u
end
```

Algorithm 13. SSSP for DAGs

Rivest, 1990, p. 532). If we restrict ourselves to positive edge weights — which is possible given the acoustic scores we use —, the better algorithm of Dijkstra (Cormen, Leiserson and Rivest, 1990, p. 527) may be used, which has a complexity of $O(|V| \log |V| + |E|)$ after some suitable modifications.

As the graphs we consider here are acyclic, we can use the even better linear ($O(|V| + |E|)$) algorithm of Lawler (Cormen, Leiserson and Rivest, 1990, p. 536). We show that method as Algorithm 13.

In the framework of this volume, the search for best paths through a graph takes place in the generation of natural language expressions. The generation (cf. Section 4.9) produces English surface representations for successfully transferred semantic descriptions. The primary result is a hypergraph of edges annotated with English utterance fragments. Within this graph we have to constantly and incrementally maintain the best path, in order to be able to output the correct combination at any point in time. Algorithm 14 shows the necessary operations. We don't need to apply the full power of incremental algorithms (Cheston, 1976) as we can induce a topological order on the graph. In particular, our case does not allow the deletion of edges, which would have to be handled using additional precautions (Ramalingam and Reps, 1992). The algorithm we present here is called once for each edge that is introduced into the graph. It maintains two values per vertex: The weight of the best path so far (b_v) and the edge that led to the vertex on the best path (r_v). We only consider vertices to the right of the inserted edge, and only carry out a re-computation if the weight for the best path was actually changed by the new edge. The generator described in Section 4.9, in fact, uses a modified version of this algorithm, as it also models the bridging of gaps in the graph. Conceptually, this is

Update single source shortest information
procedure UpdateSSSP(G, e**)**
Topological order o
Initialization (done only once)
[1] **for** each $v \in \mathcal{V}$, **do**
[2] $b_v \longleftarrow \infty$
[3] $b_{v(r)} \longleftarrow 0$

Update immediate end vertices of e
[4] **for** each $v \in \alpha_<(e)$, **do**
[5] **for** each $w \in \alpha_>(e)$, **do**
[6] **if** $b_v + w(e) < b_w$, **then**
[7] $b_w \longleftarrow b_v + w(e)$
[8] $r_w \longleftarrow e$
[9] Insert w into o

Update vertices if necessary
[10] **while** $o \neq \emptyset$, **do**
[11] $v \longleftarrow o.First()$
[12] **for** each e **having** $v \in \alpha_<(e)$, **do**
[13] **for** each $w \in \alpha_>(e)$, **do**
[14] **if** $b_v + w(e) < b_w$, **then**
[15] $b_w \longleftarrow b_v + w(e)$
[16] $r_w \longleftarrow e$
[17] Insert w into o
end

Algorithm 14. SSSP for incremental hypergraphs

done by adding virtual edges between each pair of neighboring vertices, which are annotated with a transition penalty as weight.

2.8 Summary

Word graphs are the most advanced interface between a speech recognizer and the subsequent linguistic processing. Compared to the representation of recognition results by individual word sequences, they have the advantage of being able to represent huge numbers of utterance hypotheses ($> 10^{30}$) in a very compact way. However, they are not completely free of redundancy: Identical relevant word sequences may be generated by following different paths through a graph. This offers several possibilities for reducing the size of word graphs (e.g. by reducing the graph to unique label sequences), which could in principle be applied during the computation even for incremental word graphs. However, these methods can be expensive.

An extraordinarily efficient method of reducing the effort of processing a word graph is the transition to hypergraphs, where edges carry sets of start and end vertices.

Finally, the measures used for word graph size and word graph quality are important for the comparison of different speech processing systems. We think that instead of using the density of graphs, which depends on the transliteration of the input and does not take into account the topology of a graph, a more application relevant measure for word graph size should be used: The number of operations that an ideal, fictitious parser would need to process a graph. Moreover, there may be alternatives to the usually used quality measure of word accuracy; however, we feel that using the deviation from a reference will be valid for a long time.

Chapter 3

Unification-Based Formalisms for Translation in Natural Language Processing

The formalism used for the description of linguistic knowledge is a central part of every natural language processing system. After a short introduction into unification, this chapter will provide a brief presentation of unification-based formalisms and methods that have been employed within transfer-oriented machine translation systems. Finally, we describe the formalism we used in the present system and give some details about its implementation.

3.1 Unification-Based Formalisms for Natural Language Processing

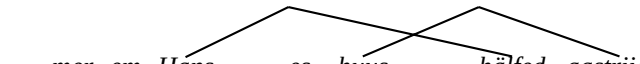
Unification-based formalisms allow for the declarative formulation of grammatical knowledge. They use a concise, generalizing representation and can cope well with partial information. The development of such formalisms has several advantages over automata-oriented approaches or methods not primarily targeted at automatic processing (for instance in the framework of generative grammar, see Chomsky, 1995).

To use automata to represent knowledge about natural languages stems from work done in the field of compiling programs written in formal programming languages — where the use of automata has been very successful. Applied to natural languages, ATNs (Augmented Transition Networks) have been the primary means used (Woods, 1973). ATNs are recursive transition networks whose edges can be annotated by conditions; moreover, registers are introduced that can store arbitrarily long values (they can even be assigned to default values). Using these augmentations, ATNs have the power of recognizing type-0 languages (Chomsky, 1959). Although some large systems have been built using ATN grammars (Kaplan, 1973), the register mechanism makes it difficult to understand the functioning of an actual network. Due to the non-monotonicity of registers, the process of an analysis can

J.W. Amtrup: Incremental Speech Translation, LNAI 1735, pp. 65-84, 1999.

not be determined from a given result. Additionally, the character of ATNs seems to be too procedural, the level of abstraction too low.

A method of description that is declarative and much more easy to follow is the use of phrase structure grammars. Early on, context-free grammars (CFGs) — equivalent to simple recursive transition networks[1] — have been used for syntactic representation of natural languages. Several classic reports on efficient parsing use context-free grammars (e.g., Earley, 1970). However, it has been argued that context-free grammars are insufficient for representing natural language adequately (cf. Sampson, 1983, or Shieber, 1985). Nevertheless, a major portion of natural language sentences can be modeled context-free, thus the existence of problematic cases alone does not seem sufficient reason for abandoning context-free grammars completely (Gazdar, 1983). A slight extension to the context-free paradigm, for example, is enough to handle classic non context-free phenomena like cross-serial dependencies in Dutch **(3.1)** (Vogel, Hahn and Branigan, 1996). The extension is necessary because in principle there might be an unlimited number of pairs of constituents that belong to each other. They are crossed and show agreement.



…	*mer*	*em*	*Hans*		*es*	*huus*		*hälfed*	*aastriiche* **(3.1)**
…	we		Hans-DAT		the	house-ACC		helped	paint

…we helped Hans paint the house.

Thus, the reason for the widespread use of unification-based formalisms and their application to the modeling of syntactic structures cannot be sought solely in their theoretical necessity. Rather, using unification renders important advantages when constructing grammars with a large coverage. Especially in comparison to context-free grammars, unification grammars allow us to write concise, general rules. For instance, the subject-verb agreement in German requires only one rule demanding that certain features of the description of both be unifiable; a context-free grammar would need one rule for each combination of possible feature values.

Moreover, feature structures are able to represent partial information, which is difficult to do in the context-free case. Again, this could be valuable for describing agreement, for example if the morphological gender marking is ambiguous.

Several unification-based formalisms allow the representation of constraints over feature structures, which guarantee the consistency of descriptions of linguistic objects during the course of a computation. This constraint-based approach is extremely powerful and in the extreme case leads to a very small set of principles and general schemata (e.g. in the case of HPSG, cf. Pollard and Sag, 1987; Pollard and Sag, 1994).

The work presented here uses a formalism which assigns a type to every feature structure. Thus, the space of potential descriptions can be further structured. The set of types gets more compact by introducing inheritance over types, which leads

[1] I.e., there is a recursive transition network for every CFG accepting the same language.

to a lattice-based representation of types and feature structures. Moreover, the unification of feature structures becomes more efficient as only feature structures with "compatible" types can be unified. The unification of types is extremely efficient using a suitable representation of the type lattice.

The origins of unification lie with Herbrand; the modern view begins with Guard (1964) and Robinson (1965). The unification algorithm by Robinson (1965) has an exponential complexity and solves the problem of unifying two terms of first order predicate logic. The result is a substitution of variables such that the two terms become identical after replacing variables with their values. Knight (1989) presents the advantages of turning to graph unification which gives up fixed arity of terms and anonymity of substructures, replacing them with named features and a coreference mechanism. The most widespread unification algorithm is by Huet (cf. Knight, 1989, p. 98), which is almost linear[2]. There are linear algorithms (Paterson and Wegman, 1978), but the constant factors of these are high, rendering them inferior to other algorithms for practical purposes (cf. Amtrup, 1992, p. 67). Interesting with regard to the work presented here are parallel implementations of unification algorithms like Vitter and Simons (1986) or Hager and Moser (1989). However, according to Knight (1989), unification is a highly sequential process that can only be noticeably speeded up by employing considerable resources. The process of unification becomes complicated if disjunctive terms are introduced (Eisele and Dörre, 1988). The problem is that disjunctions may interact with coreferences, which are not restricted to the disjunctive terms (Kaplan and Maxwell, 1989). One solution uses named disjunctions which always refer to the originally used terms during unification (Backofen, Euler and Görz, 1991).

Kay (1979) was the first one to use unification-based formalisms in natural language processing. Since then, several highly specialized theories for the representation of linguistic facts have emerged, e.g. *Lexical Functional Grammar* (LFG, Bresnan, 1982) and *Head Driven Phrase Structure Grammar* (HPSG, Pollard and Sag, 1987; Pollard and Sag, 1994). Additionally, there are more system-oriented formalisms and tools, like *Definite Clause Grammars* in Prolog (DCG, Pereira and Shieber, 1984), PATR II (Shieber, 1984) or *Unification Tree Adjoining Grammars* (UTAG, Joshi, 1985; Harbusch, 1990).

The formalisms mentioned so far are declarative in two respects: First, they describe linguistic facts without dealing with mechanisms for the solution of a specific task. Second, feature structures themselves are treated as data objects, the processing of which is externally defined. Carpenter and Qu (1995) argue, however, that feature structures can be implemented as an abstract machine. A first complete implementation of a formalism following these guidelines is given by Wintner and Francez (1995a) and Wintner (1997). The realization described here is strongly oriented towards this view.

[2]The time complexity is $O(n \cdot \alpha(n))$, $\alpha()$ being the reciprocal of the Ackermann function.

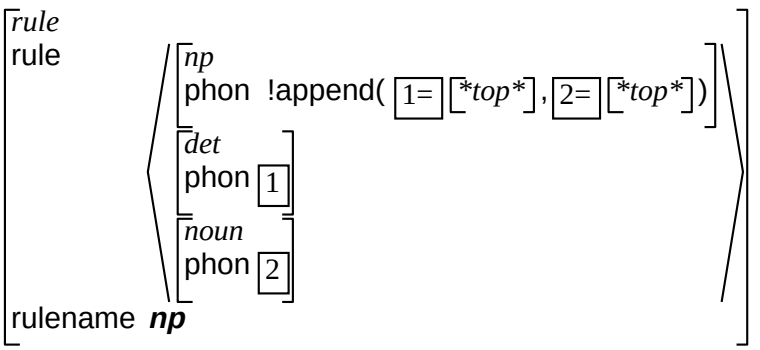


Figure 3.1. Feature structure for a simple syntactic rule

3.1.1 Definition of Typed Feature Structures with Appropriateness

In this section, the feature structures used in this monograph are defined. These definitions form the basis for the implementation of the feature structure formalism presented in Section 3.3. The goal here is to describe feature structures as in Figure 3.1[3] formally. We use a formalization following Carpenter (1992).

3.1.2 Type Lattices

Every feature structure is assigned a type. Thus, the space of feature structures is divided into desired classes. The set of types is ordered in a hierarchical way in order to further augment the representation. This augmentation is done by inducing additional structure through ordering and by using an inheritance mechanism. By adding a most general type *top* ($\top$)[4] and an inconsistent type *bottom* ($\bot$) the hierarchy becomes a lattice of types.

The partial order for the lattice is called *subsumption*. A type σ subsumes another type τ (written $\sigma \sqsupseteq \tau$), if σ is more general than τ, i.e. it is closer to $\top$. The greatest lower bound of a set of types w.r.t. the subsumption is the result of unification of types.

[3] *Italic names* denote types (**top** denotes $\top$, see below), sans serif names denote feature names, ***bold sans serif*** denote strings.

[4] In this work, we use a model-theoretic view. $\top$ is the most general type since it contains all extensions.

Definition 3.1.1 (Type lattice).
Let $D = (T, \sqsupseteq)$ be a partial order, T being a set of type symbols. Let $\sqsupseteq$ be the partial order relation over T, i.e. $\sqsupseteq$ is reflexive, antisymmetric and transitive.

Let $D \subseteq T$ be a subset of types. An element $s \in T$ is called least upper bound (supremum) of D, written $\sqcup D$, if:

$$\bigwedge_{x \in D} \quad s \sqsupseteq x \tag{3.1}$$

$$\bigwedge_{s\prime \in T} \quad (\forall x \in D : s' \sqsupseteq x) \Longrightarrow s' = s \tag{3.2}$$

An element $i \in T$ is called greatest lower bound (infimum) of D, written $\sqcap D$, if:

$$\bigwedge_{x \in D} \quad x \sqsupseteq i \tag{3.3}$$

$$\bigwedge_{i\prime \in T} \quad (\forall x \in D : x \sqsupseteq i') \Longrightarrow i' = i \tag{3.4}$$

The extension $D\prime = (T \cup \{\top := \sqcup T, \bot := \sqcap\emptyset\}, \sqsupseteq)$ forms a lattice, the type lattice of types from T. *$\sqsupseteq$ is called* subsumption relation, *the operation of finding the infimum $\sqcap D$ of a subset of types is called* type unification.

3.1.3 Feature Structures

We can now define feature structures as directed graphs with edge and vertex labels and a designated root. The nodes of a feature structure graph are annotated with types, the edges bear feature names as labels. Cycles are explicitly allowed and denote coreferences.

Definition 3.1.2 (Feature structures, Paths).
Let $D = (T, \sqsupseteq)$ be a type lattice, $Feat$ a finite set of feature names, $Feat \neq \emptyset$.

A feature structure *F is a quadruple $F = \langle Q, \overline{q}, \theta, \delta \rangle$, iff the following holds:*

- *Q is a finite set of vertices, $Q \neq \emptyset$.*
- *$\overline{q} \in Q$ is the root of the feature structure.*
- *$\theta : Q \longrightarrow T$ is a total function, mapping vertices to types.*
- *$\delta : Q \times Feat \longrightarrow Q$ is a partial function, which determines feature values.*

The definition of δ is extended by allowing the traversal of more than one feature at a time. Let $Path := Feat^*$ be the set of paths over feature names, let ϵ be the empty path without any feature names. We define additionally

$$\delta(q, \epsilon) \quad := \quad q, \textit{ and} \tag{3.5}$$

$$\delta(q, f\pi) \quad := \quad \delta(\delta(q, f), \pi), \textit{ where } q \in Q, f \in Feat, \pi \in Path \tag{3.6}$$

The function δ constitutes the edges of the graph representing the feature structure. The vertices in Q have to be reachable from $\overline{q}$ by repeatedly applying δ, i.e. the root of the graph is unique.

It may be the case that δ is not injective, i.e. a vertex of the graph representing a feature structure may have two edges incident to it ($\exists q1, q2 \in Q \exists f1, f2 \in Feat : (q1, f1) \neq (q2, f2) \wedge \delta(q1, f1) = \delta(q2, f2)$). In this case, we say that two feature values are coreferent, depicting this fact visually by co-indexed boxes (cf. Figure 3.1). This corresponds to the use of identical variables in logic programming languages. There are cycles in a feature structure if one of the coreferring vertices can be reached from the other ($\exists \pi \in Path : \delta(q1, \pi) = q2$).

In analogy to type subsumption, we can now define subsumption over feature structures. A feature structure F subsumes another feature structure F' if F is more general than F', thereby preserving all structural properties of F in F'. This definition allows us to estimate which of two feature structures contains more information, in case partial information is represented.

Definition 3.1.3 (Subsumption of feature structures).
Let $F = \langle Q, \overline{q}, \theta, \delta \rangle$ and $F\prime = \langle Q\prime, \overline{q}\prime, \theta\prime, \delta\prime \rangle$ be two feature structures over the type lattice $D = (T, \sqsupseteq)$ and the set of feature names $Feat$. We say that F subsumes *the feature structure $F\prime$, written $F \sqsupseteq F\prime$, iff there is a mapping $h : Q \longrightarrow Q\prime$, which fulfills the following conditions:*

- *The root of F is mapped to the root of $F\prime$:*

$$h(\overline{q}) = \overline{q}\prime \tag{3.7}$$

- *The type of each vertex in F subsumes the type of the corresponding vertex in $F\prime$:*

$$\bigwedge_{q \in Q} \theta(q) \sqsupseteq \theta\prime(h(q)) \tag{3.8}$$

- *The edges in F are also present in $F\prime$:*[5]

$$\bigwedge_{q \in Q} \bigwedge_{f \in F} \delta(q, f)_{def.} \Longrightarrow h(\delta(q, f)) = \delta\prime(h(q), f) \tag{3.9}$$

The unification of feature structures is now also defined in analogy to the unification of types. The unification of two feature structures is the smallest feature structure that is subsumed by the argument structures. This view is made possible by having the subsumption relation as partial order over feature structures. The process of unification begins with the root vertices of the two operand feature structures. The root vertex of the result is assigned to the type unification of the two originating vertices. Following that, vertices that are reachable by following edges bearing the same feature name are identified, which leads to the construction of a result vertex, the type of which is again the result of the type unification of the operand vertices.

[5] We abbreviate $R(x, y)_{def.}$, is $R(x, y)$ is defined.

The unification ends if all vertices have been considered, and fails if the unification of types fails at any given point (i.e. gives $\perp$). Formally, the definition in Carpenter (1992) uses equivalence classes.

Definition 3.1.4 (Equivalence class, Quotient set).
Let $\equiv$ be an equivalence relation, i.e. a transitive, reflexive, symmetric relation over a set X. Then,

$$[x]_{\equiv} = \{y \in X \mid y \equiv x\} \tag{3.10}$$

is called the equivalence class of x over $\equiv$.

The set

$$X/_{\equiv} = \{[x]_{\equiv} \mid x \in X\} \tag{3.11}$$

is called quotient set of X over $\equiv$.

The definition of the unification of two feature structures defines an equivalence relation that exactly fulfills the requirements stated above.

Definition 3.1.5 (Unification of feature structures).
Let $F = \langle Q, \overline{q}, \theta, \delta\rangle$ and $F\prime = \langle Q\prime, \overline{q}\prime, \theta\prime, \delta\prime\rangle$ be two feature structures over the type lattice $D = (T, \sqsupseteq)$ and the set of feature names $Feat$. Also, let $Q \cap Q\prime = \emptyset$.[6] Let the equivalence relation $\bowtie$ be the smallest equivalence relation, such that

$$\overline{q} \bowtie \overline{q}\prime, \text{ and} \tag{3.12}$$

$$\delta(q, f) \bowtie \delta\prime(q\prime, f), \text{ if both are defined, and } q \bowtie q\prime \tag{3.13}$$

The Unification of F and $F\prime$ is then defined as

$$F \sqcap F\prime = \langle (Q \cup Q\prime)/_{\bowtie}, [q\prime]_{\bowtie}, \theta^{\bowtie}, \delta^{\bowtie}\rangle, \tag{3.14}$$

where

$$\theta^{\bowtie}([q]_{\bowtie}) = \sqcap\{(\theta \cup \theta\prime)(q\prime) \mid q\prime \bowtie q\} \tag{3.15}$$

and

$$\delta^{\bowtie}([q]_{\bowtie}, f) = \begin{cases} [(\delta \cup \delta\prime)(q, f)]_{\bowtie} & \text{, iff } (\delta \cup \delta\prime)(q, f) \text{ is defined} \\ \text{undefined} & \text{, else} \end{cases} \tag{3.16}$$

if all type unifications in $\theta^{\bowtie}$ exists. Else, $F \sqcap F\prime$ is undefined.

The unification of two feature structures is again a feature structure if the unification is defined (cf. Carpenter, 1992, p. 47). In particular, it can be shown that the unification conjunctively assembles information from both argument structures. Features that are only present in one of the arguments are transferred into the result by virtue of $\bowtie$ being reflexive.

[6] The unification constructs the new feature structure by a set union of the original vertex sets under certain circumstances. Thus, we demand an empty intersection that can always be reached by using alphabetic variants (cf. Carpenter, 1992, p. 43).

Appropriateness. The definition of feature structures as it is now uses the type hierarchy during unification; however, it does not restrict the use of features at vertices of given types. But this is exactly one property demanded in several formalisms (e.g. HPSG, cf. Pollard and Sag, 1987) in order to distinguish invalid features from features that happen to have no value. Moreover, the restriction of admissible features for types allows a representation of feature structure nodes having a fixed size. This property will turn out to be relevant for the implementation of the formalism described below.

Carpenter (1992) defines an *appropriateness function* as a partial function which states which features are admissible for a specific type and what types the values of those features must have.

Definition 3.1.6 (Appropriateness).
Let $D = (T, \sqsupseteq)$ be a type lattice and $Feat$ a finite, nonempty set of feature names. A partial function $Approp : T \times Feat \longrightarrow T$ is called appropriateness function, if the following properties hold:

- *Each feature name is introduced at exactly one position in the lattice:*
  For each feature name $f \in Feat$ there is a most general type $Intro(f) \in T$, such that
  $Approp(Intro(f), f)$ is defined.
- *The appropriateness is downward closed:*
  If $Approp(\sigma, f)$ is defined and $\sigma \sqsupseteq \tau$, then $Approp(\tau, f)$ is also defined and $Approp(\sigma, f) \sqsupseteq Approp(\tau, f)$.

In the application of a formalism with appropriateness only those feature structures are considered that are well-formed according to the requirements of appropriateness. This is done by restriction to well-typed feature structures.

Definition 3.1.7 (Well-typed feature structures).
A feature structure $F = \langle Q, \overline{q}, \theta, \delta \rangle$ is called well-typed iff

$$\bigwedge_{q \in Q} \bigwedge_{f \in Feat} \delta(q, f)_{def.} \longrightarrow Approp(\theta(q), f)_{def.} \wedge Approp(\theta(q), f) \sqsupseteq \theta(\delta(q, f)). \tag{3.17}$$

In this work, we only use well-typed feature structures. However, the restrictions defined by the appropriateness functions are weaker than stated by Definition 3.1.6. We do not demand that there be a unique introducing type $Intro(f)$ for each feature name. This inhibits the use of *type inference* and thereby makes classification of feature structures according to the type lattice impossible, but we will not need those operations. A further property of well-formedness is given by the notion of completely well-typed feature structures. Along with being well-typed, it is additionally demanded that each admissible feature also have a value.

Definition 3.1.8 (Completely well-type feature structures).
A feature structure $F = \langle Q, \overline{q}, \theta, \delta \rangle$ *is called* completely well-typed *iff* F *is well-typed and*

$$\bigwedge_{q \in Q} \bigwedge_{f \in Feat} Approp(q, f)_{def.} \longrightarrow \delta(q, f)_{def.} \tag{3.18}$$

3.1.4 Functions as Values of Features

Some feature structure formalisms allow the formulation of functions and relations over feature values. For instance, Emele and Zajac (1990) use the annotation of conditions for type definitions to introduce recursive functions. HPSG (Pollard and Sag, 1994) defines relations within feature structures that are always kept. In that case, they are used (among others) to introduce list operations (e.g. for phonological representations or operations over subcategorization information).

In the present implementation, we do not use complete relations of conditions for feature structures. In particular, the annotation of conditions inhibits a fine-grained control over the process of unification, as the checking of constraints may introduce subsequent unifications. However, we define functions on feature structures, which can be evaluated independently of the unification as such. These function calls have a more procedural character (cf. Section 3.3.2).

3.2 Unification-Based Machine Translation

Almost immediately after Kay (1979) introduced unification-based formalisms into the processing of natural languages, their appropriateness and value for the transfer phase in machine translation became obvious. Kay (1984) proposes the modeling of transfer knowledge within the *Functional Unification Grammar* (FUG), other authors use similar methods. It turns out that one needs two notions of equivalence between feature structures, one of which is the identity defined in terms of coreference. The other one represents the application of subordinate transfer rules, thereby modeling compositional transfer. In many approaches this is done by providing a specific form of transfer rules, together with a processing mechanism taking care of these rules.

Kaplan *et al.* (1989) use the correspondences already provided in the framework of LFG. Correspondences model relations between different levels of the linguistic representation of an utterance. There are already two correspondences in the original theory: Φ mediates between the constituent structure (*c-structure*) and the level of syntactic functions (*f-structure*), the relation σ describes the additional mapping to the level of semantic representations (cf. Kaplan and Bresnan, 1982).

In order to be able to use LFG for transfer problems, Kaplan *et al.* (1989) introduce an additional relation, τ, that describes relationships between source and

beantworten V
(↑ PRED) = 'beantworten ⟨ (↑ SUBJ) (↑ OBJ)⟩'
(τ ↑ PRED FN) = repondre
(τ ↑ SUBJ) = τ (↑ SUBJ)
(τ ↑ AOBJ OBJ) = τ (↑ OBJ)

Figure 3.2. A transfer rule in LFG style

target language. By using a functional composition of the three relations Φ, σ and τ, transfer rules on different levels of abstraction can be formulated.[7] In the rule depicted in Figure 3.2 there are compositions of ↑[8] and τ in distinct order. The order of application defines whether elements of a relation are searched for on the source language side of an utterance or the target language side. In the present example, the translation of the verb "beantworten" (answer) into French requires the insertion of the preposition "à" in front of the object. This is ensured by the additional path AOBJ in the rule.

Der	*Student*	*beantwortet*	*die*	*Frage.*
Le	étudiant	répond	la	question.

"L'étudiant répond à la question."

(3.2)

A number of approaches use different paths within feature structures to separate information belonging to different languages. These approaches include Noord (1990), Carlson and Vilkuna (1990) and Morimoto *et al.* (1992). Zajac (1992), for example, uses a typed feature structure formalism, which leads to a reduction of the number of transfer rules if inheritance within the type lattice is used. Zajac uses two features, ENG and FR, to represent structures in English and French. Coreferences across the language border are allowed.

Recursive transfer is expressed by conditions internal to the formalism (TFS, cf. Emele and Zajac, 1990), thus no further external mechanisms to handle this are necessary. A simple translation

A	*student*	*falls.*
Un	étudiant	tombe.

(3.3)

[7] However, this elegant solution cannot cope with embeddings (cf. Sadler and Thompson, 1991).

[8] ↑ is an abbreviation for the immediate dominance.

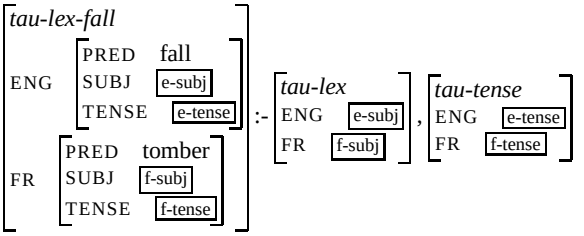


Figure 3.3. A transfer rule in TFS style

```
Label
    NP-def
Source
    <* cat>     = NP
    <* def>     = DEF
    <* num>     = ?Num
    <* head>    = ?head1
Target
    <* cat>     = NP
    <* def>     = DEF
    <* num>     = ?Num
    <* det lex> = 'the'
    <* head>    = ?head2
Transfer
    ?head1     <=> ?head2
```

Figure 3.4. A transfer rule by Beskow

may be achieved by applying the transfer rule in Figure 3.3. Note that tau $\sqsupseteq$ tau-lex $\sqsupseteq$ tau-lex-fall holds.

The different aspects a transfer rule shows are made explicit in the approach of Beskow (1993) by dividing the knowledge within a rule into four different domains. Consider the rule in Figure 3.4 as an example, which translates definite noun phrases from Swedish to English.

The four parts of a rule are:

- A unique name for the rule (`Label`).
- A source language structure that checks the applicability of a rule (`Source`). A transfer rule is suitable for the translation of an utterance fragment if the structural description can be unified with the `Source` part of the rule. The star symbol (‘`*`’) denotes the root of the graph, coreference is expressed by variable binding (names of variables begin with ‘`?`’).

- A target language structure (`Target`) that can be used either for the verification or the construction of a translation (the model can be applied to both cases).
- A list of embedded transfer equations (`Transfer`), that describe recursive transfer. The individual equations are iteratively accounted for to achieve the translation of a complex structure.

Most of the aforementioned systems separate the control part of transfer from the declarative description of contrastive knowledge about the languages in question. It should be noted that in addition to the identity provided by coreferences — part of the standard repertoire of most feature structure formalisms — the relation of subsequent, recursive transfer has to be expressed. This is mostly done by special feature names within feature structures that are evaluated by the transfer mechanism. Only formalisms capable of annotating conditions to feature structures are able to carry out a sub-aspect (the automatic evaluation of recursive transfer) internal to the formalism (Zajac, 1992). However, one loses the possibility to establish specific control strategies that rule over the applicability of subsequent transfer equations.

3.3 Architecture and Implementation of the Formalism

We developed a feature structure formalism with types and appropriateness used in the processing mechanisms and system modules of the speech interpreting application described in this monograph. The implementation realizes the objects described in the preceding sections, as well as the operations defined on them; thus, it strongly resembles parts of ALE (Carpenter and Penn, 1998), subject to the restrictions already mentioned, e.g. related to the introduction of types.

The current implementation follows partly the work of Carpenter and Qu (1995) and Wintner and Francez (1995a), who propose an interpretation of feature structures as abstract machines, similar to the *Warren abstract machine* for Prolog (Warren, 1983). A first complete implementation of such a machine is given by Wintner (1997). We follow this machine-oriented view only to the extent of using consecutive memory locations for the representation of feature structures, as was already done in Amtrup (1992) for untyped feature structures. The motivation for this model, which is in contrast to usual interpretations using a distributed representation schema with heavy pointer support, is to implement a formalism suitable for use in different parallel machine environments. The underlying assumption is that — given a sufficiently fine-grained system — the fraction of feature structures that have to be communicated between different components of a distributed system is high. If one uses a conventional implementation of a feature structure formalism, this means that highly discontinuous representations have to linearized before this linear form is communicated. In most cases, the representation transmitted is a character string without any internal structure.[9] On the receiving end of the communication link, the

character string is parsed in order to again create an internal representation of the desired feature structure that can be used further.

In contrast to this schema, the feature structures defined with the formalism used here are invariant to memory location changes, and invariant to interpretation in a different address space on a different machine. This makes the communication of feature structures much simpler, as the communication of a feature structure only involves communicating a consecutive range of memory locations, without the need of the pre- and postprocessing just mentioned. The disadvantage, however, lies in the fact that unification routines have to take the special kind of encoding into account. This involves tracking some bookkeeping information and inhibits the sharing of partial structures across feature structures (Billot and Lang, 1989). But in spite of these drawbacks, efficient algorithms and representation mechanisms can still be used.

The formalism described here — in contrast to ALE, TFS (Zajac, 1991) and CUF (Dorna, 1992) — is not realized as an extension to Prolog, but is implemented as a separate module in C++. This is due to the fact that we required the possibility of employing the formalism on parallel architectures. The use of C++ guarantees an easy way of porting the module to several existing multiprocessor architectures, since C++ is provided there without exception, while typical AI-languages such as Lisp or Prolog are not widely available in the parallel computer community.[10] It follows that the MILC system, developed within the scope of this work, is not only suitable for inter-modular parallelism, but also for intra-modular parallelism.

The formalism should be just powerful enough to model all desirable properties, without introducing too much overhead for rarely used parts. Hence, this implementation does without negation and complex disjunction[11] and without the annotation of constraints on feature structures in the style of TFS. Disjunction is only defined for atomic types. We use it mainly to describe alternatives on the atomic level that are not captured by the type lattice (e.g. to represent sets of different semantic concepts).

The control over the processing of different objects of the formalism is left exclusively to the user. This gives the developer of modules the largest possible freedom about the choice and implementation of control strategies, which are com-

[9] In rare cases, there are communication systems capable of handling certain complex data types. For example, the communication system ICE (Amtrup, 1994a; Amtrup, 1995b; Amtrup and Benra, 1996), which has also been used in the work presented here, is able to transmit the semantic structures used in the Verbmobil project (Verbmobil Interface Terms, VITs). They are used to represent flat semantic structures. But even in this case, an analysis of the data objects being transmitted is necessary, the only difference being that this mechanism is handled transparently.

[10] There are exceptions to this rule, the most prominent being the *Connection Machine* (Hillis, 1985), which supports a vector-parallel List (Steele and Hillis, 1986), as well as some implementations for Transputer systems (cf. Kesseler, 1990).

[11] In the framework of MILC, we express complex disjunction by using different alternative feature structures. This enables a fine-grained control of the evaluation that would not be possible if disjunction was formulated within the formalism.

pletely immanent to the formalism in other systems, rendering them almost unaccessible from outside. The control mechanism for the current formalism not only gives control over the execution of unification of different feature structures, but additionally, the evaluation of parts of the unification mechanism (the evaluation of function calls) can be separately activated.[12]

The current implementation distinguishes three different sets of functions that successively offer an extended functionality, but also demand successively higher memory usage. The motive for this tripartite implementation is again to provide a system suitable for employment on a parallel machine. To achieve this, it is reasonable to keep the runtime memory resources needed as small as possible, since parallel machines usually have restricted resources per processing node. The restrictions concern two areas:

- The accessibility of names. While compiling feature structures from an external, readable format to an internal representation that is oriented towards efficiency of algorithms, names are converted to unique numeric identifiers. After this conversion is done, it is no longer necessary to store all possibly used names in memory.
- The direct accessibility to the type lattice. The elements of the type system are internally represented as bit vectors. The data structures and program routines used for this compilation step are irrelevant for the application of a system. Thus, they can be excluded from a runtime version without any loss of functionality.

By successively excluding the properties just mentioned, three different levels of the feature structure machine emerge:

- **Restricted feature structures**. This module only contains input/output functions for feature structures that are already represented in the internal format. Additionally, all necessary unification routines are provided. Names of types, features and values can not be printed in a readable format. The foremost area of use of this variety is for processing modules within a parallel machine implementation. The extensive output of feature structures is usually not needed, since almost none of the processing nodes have a direct communication link to the outer world. Instead, typically there is a dedicated module functioning as an interface between the parallel system and the user. Only in that module does the user need input/output functions to communicate with the system. All other means of communication should happen using the internal format.
- **Standard feature structures**. Apart from the functionality provided by restricted feature structures, this module offers functions for input and output of feature structures in human-readable form. This requires functions for compiling external feature structures into the internally used format, as well as printing routines. If components that use other formalisms are integrated, this set of functions should also be used.
- **Extended feature structures**. In addition to the functions already mentioned, this library contains compilation functions allowing the reconstruction of type

[12] The $\mathcal{TDL}$ system (Krieger, 1995), a very extensive system for the description of constraint-based grammars, allows for individual calls of type expansion.

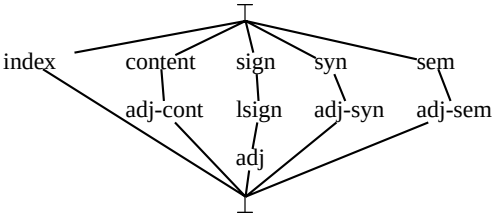


Figure 3.5. A small part of a type lattice

```
type sign isa *top* intro phon:*top* syn:syn sem:sem.
type lsign isa sign.
type syn isa *top*.
type sem isa *top* intro cont:content.
type content isa *top*.
type adj isa lsign intro sem:adj-sem syn:adj-syn.
type adj-sem isa sem intro cont:adj-cont.
type adj-cont isa content intro index:index cond:*top*.
type adj-syn isa syn.
type index isa *top*.
```

Figure 3.6. One part of the structure of lexical items

lattices. This requires an extended representation of the name space in the system and several other procedures for the compilation of lattices and the analysis of lattice definitions.

3.3.1 Definition and Implementation of Type Lattices

The formalism at hand uses typed feature structures. Figure 3.5 shows a small excerpt of the type definitions for lexical entities as an example of a type lattice. The definitions leading to that structure are presented as Figure 3.6. This example clearly shows how certain features are introduced into the hierarchy at an early point. Each lexical sign contains features defining the phonological representation (phon), the syntactic features (syn), as well as semantic properties (sem). Types at lower levels of the hierarchy further specialize the values of certain features. For instance, the semantic content of adjectives (sem|cont) is required to be of type adj-cont. Restrictions of this type can be accompanied by the introduction of additional features.

Such a definition of a hierarchy of types results in a lattice (cf. Definition 3.1.1). We always implicitly define a type `*top*` ($\top$), which is the root of the lattice.

Table 3.1. The syntax for type lattices

Typelattice	::=	Entry$^+$
Entry	::=	Typedefinition | Subtypedefinition **.**
Typedefinition	::=	**type** Typename **[**Supertype**]** **[**Appropriateness**]**
Supertype	::=	**isa** Typename$^+$
Appropriateness	::=	**intro** Approp-Feature$^+$
Approp-Feature	::=	Featurename **:** Typename
Subtypedefinition	::=	**types** Typename **'are'** Typename$^+$
Typename	::=	Symbol
Symbol	::=	Character$^+$
Character	::=	*all printable characters except* **[**␣**.:()[]**⟨ ⟩**$,!]**

Further types are defined as subtypes of already existing types. For each type, admissible features can be introduced, the definition of the appropriateness is carried out in parallel to the construction of the type lattice. The syntax used to formulate type lattices is described in Table 3.1. In order to allow for a simpler method of creating the necessary relations, an alternative syntax was introduced, defining the subtypes of a super-type.

Note that we do not demand that a feature is introduced at a unique position within the lattice (see above).

The internal representation of types uses bit vectors. This corresponds to the compact encoding for type lattices given in Aït-Kaci *et al.* (1989). Storing types as bit vectors has several advantages, among them:

- The representation of types within a hierarchy is very compact. The type lattice used in this work contains 345 types, yet uses only nine integer values (288 bits) to store a type. An explicit representation of the partial order over types is not needed. Feature structures use references to the lattice and do not store the actual bit vectors.
- The operations on types can be implemented very efficiently, the effort for computing the unification of two types becomes almost constant. Checking the subsumption relation between two types and the unification of types can be reduced to the calculation of a bitwise *AND* between bit vector representations.

3.3.2 Definition and Implementation of Feature Structures

The feature structures that can be used by the formalism of the current system are typed feature structures with appropriateness. Figure 3.1, which we repeat here as Figure 3.7, shows an example of such a feature structure.

The implementation of feature structures shows some special properties:

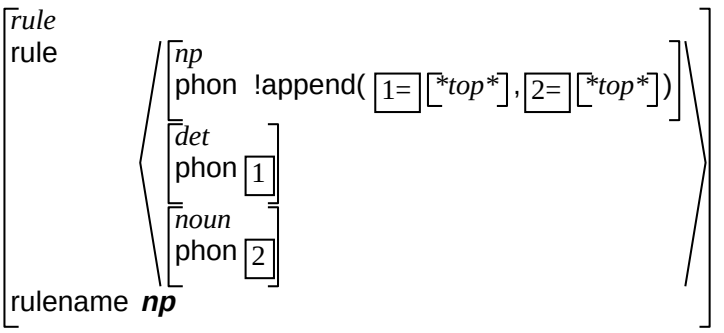


Figure 3.7. Feature structure for a simple syntactic rule

- The availability of expressing atomic disjunction.
  This is possible without referring to the type lattice, thus the user may specify disjunctions of atomic types which have no common super-type except *top*.
- The direct representation of lists.
  Usually, lists of feature structures are built by recurring to the type lattice. The type *list* is partitioned into two subtypes: the empty list (*elist*) and the non-empty list (*nelist*). The non-empty list has two appropriate features, head and tail. The head is appropriate for type **top**, the tail element of the list is appropriate for *list*. Using recursion over the rest of a list, arbitrary long lists can be constructed. Implementing lists directly as objects of the formalism — the choice we took here — gives the advantage that unification routines know about the structure of lists and can use that knowledge to increase efficiency. Moreover, this strategy makes it easier to implement polymorphic lists, which can define constraints over the type of their elements (this is a property we do not yet use).
- The introduction of function calls.
  Instead of having a feature structure as a value for a feature, that value can be assigned as the result of a function evaluation. In the current system, this possibility outside the scope of a conservative formalism is mainly used to concatenate different phonological values.
  This extension is destructive in the sense that we do not define relations that are constantly checked during the construction of feature structures, like HPSG does. Rather, the unification of two feature structures is carried out in two phases: First, function calls are neglected and a conventional unification is computed. After that has been done, the user may decide to evaluate function calls within feature structures. A function application that fails is treated as a failure in unification. If, however, the function returns successfully, the return value is inserted replacing

Table 3.2. The syntax of feature structures

TFS	::=	**[**Typename**]** FS | List | OpenList | Pair | Coref | CorefDef | Nil | Atom | Set | String | Function
FS	::=	**[(**FeatureName TFS**)*** **]**
List	::=	⟨ FS* ⟩
OpenList	::=	⟨ FS* **...** ⟩
Pair	::=	⟨ FS **.** FS ⟩
Coref	::=	**%**Name
CorefDef	::=	**%**Name**=** FS
Nil	::=	**nil**
Atom	::=	TypeName
Set	::=	**(** TypeName* **)**
String	::=	**"** CharacterSequence **"**
Function	::=	**!**FunctionName**(** Argument* **)**
Typename	::=	Symbol
Symbol	::=	Character+
Character	::=	*all printable characters except* **[**␣**.:()[]**⟨ ⟩**$,!]**

```
typedef struct FNode {
  int sort;           // Sort of node (Atom, Conj, ...)
  int value;          // Value of this node (Type, ...)
} FNode;
```

Figure 3.8. Definition of vertices within feature structures

the function call. Once all function calls have been evaluated, the result is a feature structure free of side effects.

All feature structures comply with the syntax given in Table 3.2. It is redundant to provide the type of a feature structure as long as it can be derived from the appropriateness definition in the current context. A classification of feature structures cannot be computed, however, since we do not demand a unique introductory type for each feature, and we only model well-typed feature structures (see Definition 3.1.7), and not totally well-typed feature structures (see Definition 3.1.8). Features required by the appropriateness definition may be empty.

The internal format of feature structures in memory consists of an array of pairs, which represent vertices of the feature structures. Each pair consists of a `sort`, which defines the semantics of the immediately following `value` field (cf. Figure 3.8). References within a feature structure are implemented as pointers relative to the beginning of a feature structure. Thus, we guarantee a consecutive representation which enables an efficient communication of feature structures, as already mentioned.

Table 3.3. Internal representation of a feature structure

Node No.	Node Sort	Node Value	Description
0	0	12	Pointer to function definitions
1	FST_TYPE	rule	Topmost node in the feature structure
2	FST_FEAT	6	Pointer to the feature rule
3	FST_FEAT	4	Pointer to the feature rulename
4	FST_STRING	3	String of length 3
5	1852833792	0	String representation
6	FST_LIST	0	The feature rule is a list
7	FST_FIRST	9	Pointer to the head of the list
8	FST_REST	20	Pointer to the rest of the list
9	FST_TYPE	np	Left side of the rule
10	FST_FEAT	11	Pointer to the feature phon
11	FST_EMPTY	0	phon will be filled by a function call
12	FST_LIST	0	Function definitions begin here
13	FST_FIRST	15	Pointer to the first function
14	FST_REST	19	Pointer to further functions
15	FST_FUNC	1	Function *append()*
16	FST_ARG	11	Pointer to the result
17	FST_ARG	25	Pointer to the first argument
18	FST_ARG	31	Pointer to the second argument
19	FST_EMPTY	0	No more function definitions
20	FST_LIST	0	Continuation of the rule definition
21	FST_FIRST	23	Pointer to the second element of the rule
22	FST_REST	26	... and further elements
23	FST_TYPE	det	Definition of the determiner
24	FST_FEAT	25	Pointer to the feature phon
25	FST_TYPE	*top*	Empty feature structure
26	FST_LIST	0	Continuation of the rule definition
27	FST_FIRST	28	Pointer to the third element of the rule
28	FST_REST	32	
29	FST_TYPE	noun	Definition of the noun
30	FST_FEAT	31	Pointer to the feature phon
31	FST_TYPE	*top*	Empty feature structure
32	FST_NIL	0	End of the list of rule elements

Table 3.3 shows a feature structure in internal format. It corresponds to the rule shown in Figure 3.7. During the execution of the unification algorithm further properties have to be stored, e.g. to handle information for the union-find algorithm used in the unification algorithm by Huet (cf. Knight, 1989, p. 98).

3.4 Summary

Using a unification-based formalism within a system for processing natural language renders considerable advantages. The declarative method of describing linguistic knowledge is far better than other approaches. Moreover, there are now efficient algorithms to compute the unification of feature structures, which are also available as an abstract machine implementation.

Such formalisms are suitable for the transfer stage within a machine translation system, as several examples and case studies show. In this chapter, we have shown the concept and implementation of the feature structure formalism used in this monograph. It realizes complex typed feature structures with appropriateness. The representation of feature structures in internal format is very compact and well suited for the application in distributed systems due to the position-invariant encoding.

Chapter 4

MILC: Structure and Implementation

This chapter describes the architecture and implementation of the MILC system (Machine Interpretation with Layered Charts).

MILC is an incremental translation system for spoken language which takes into account the principles of integration and uniformity. The motives and essential features have been described at length in Chapter 1, and thus will only be briefly reviewed here:

- The main motivation and the predominant property of MILC is the incremental operation that is realized throughout all components. An important goal of the implementation is to provide an experimental platform for further research and studies in architectonic properties of NLP systems.
- The distribution of components over different processes or computers prevents the existence of a global system state. However, the union of information present in all components is consistent at every point in time and shows an almost current state of the system[1]; the implementation using layered charts that is described below facilitates this.
- The introduction of layered charts establishes an integrated way of processing simultaneously. Each component is in principle able to communicate with any other component. There is no need to design and implement special interfaces for the exchange of data. This is due to the fact that all information present in the system is coded as edges of the layered chart with all its components.
- The use of a uniform formalism in as many components as possible further simplifies the communication among components. The complex feature formalism described in Chapter 3 is the basis for processing in all the components that have been implemented in the framework of MILC.

In this chapter, we will first introduce and describe the notion of a layered chart and compare it to other approaches. Then, the communication infrastructure is presented which guarantees the successful operation of MILC within a distributed environment. After a short overview of the system architecture of MILC we will shed

[1] There is no global state in a distributed system that could be known to any single component. What is meant here is that the component computing the union receives information from the recent history of components.

J.W. Amtrup: Incremental Speech Translation, LNAI 1735, pp. 85-154, 1999.

light on the role and properties of the individual modules present in the system. The chapter closes with a description of further possible extensions and modifications.

4.1 Layered Charts

Time is the most important structural criterion for characterizing language. Language is linear in time. In the case of written language the ordering is given by the position of words and the spatial relationship between words. The strict linear order can be diminished by the author (by applying a higher-level order using paragraphs or other structural properties) who is able to introduce additional meaning into a text by using stylistic means. The reader, on the other hand, is able to restructure the text by browsing back and forth through a text or by not completely reading it, but scanning through it. However, if we assume spoken language as is done in this work, the linearity is almost completely valid.[2]

The underlying reason for this is, of course, that speech is produced as sound pressure that changes over time. The first stage of word recognition systems takes this into account by computing spectral parameters within time windows that are shifted throughout a complete utterance. The parameters are converted into feature vectors that encode relevant properties of the speech signal within the current window. These vectors are usually used in automata-based algorithms to search for words. Words are defined by models that consist of sequences of feature vectors representing the individual words, and words are recognized by measuring the distance between the model vectors and the actual vectors present in the input signal. Nowadays, HMM-based methods seem to be the most widely used (cf. Hauenstein, 1996). We will not delve into this early stage of machine recognition of spoken language, but it seems important to point to the shape that the input has in the dimension of time as early as possible.

Within the framework of speech processing presented here it is also evident that there is the possibility of connections between models for speech recognition and language processing that extend over the mere delivery of word hypotheses from a word recognizer to the linguistic modules. A tighter integration of speech and language processing is desirable, which would accomplish an augmentation of word recognition on the basis of linguistic facts. The extension in time of the input (in form of feature vectors on the lowest level) is essential for such an integration. We will come back to this viewpoint in Section 4.11.

The initial input to the MILC system is given by the result of an automatic word recognition process. The result consists of a set of word hypotheses, which denote words possibly spoken during a certain interval in time. We construct a word graph from these results as defined in Definition 2.2.2; more precisely, a hypergraph of word hypotheses is created. For the time being, the exact shape of the graph is irrelevant. If this graph is not viewed as *word* graph, but as *interpretation* graph

[2] We abstract away from technical means, e.g. by fast forwarding a tape.

that allows for a compact representation of possibly competing interpretations of intervals of the underlying speech signal, then it is possible to view each word hypothesis as an explanation of certain features of the input during a specific interval. On such a low level this is trivial, a word graph being specifically designed to enable that view. A more abstract level allows the syntactic analyses, semantic descriptions and even surface hypotheses in a different language to be taken as hypotheses over certain linguistic properties of parts of the input signal.

This abstract view is the first step towards the construction of layered charts. It allows the introduction of a graph structure for the representation of results.

The simplest approach to storing the results obtained during the course of processing would be to assume a monotonic augmentation of information, which rests on a uniform representation of these results. Using this method, the annotations on edges of the chart always contain all the necessary information that could be used during further processing. Moreover, it is possible to use a uniform processing mechanism for information structures, like it was done in KIT-FAST (Weisweber, 1992; Weisweber, 1994). However, this processing schema requires that all information present within an annotation to be taken into account, even if it is not completely used in a specific module; at least, irrelevant parts of the data have to be actively ignored. All information must be transmitted along information paths that are usually equipped with only a restricted bandwidth.

The goal of an efficient method of representation and storage is, consequently, to make only the necessary parts of information available for each individual component; only those parts which are essential for its computation. This type of *information hiding* leads to a much simpler structure of components. Additionally, the efficiency of computations may grow. Systems grounded in the notion of a *blackboard* (Hayes-Roth, 1995; Engelmore and Morgan, 1988) implement such an approach to information control, cf. e.g. Hearsay II (Erman *et al.*, 1980). All results are typed and stored centrally. Thus, a component only receives information that is relevant for the processing at hand, without being able to recognize the structure of the blackboard in its entirety or to know what data is contained in the blackboard. As far as the implementation of a single component is concerned, this type of storage of information is ideal. However, one has to notice that the employment of a blackboard to store data centrally always implies the use of a centralized control strategy. On one hand, this is positive, since the process controlling the blackboard is able to monitor the order in which data elements are processed; it can even establish a preference order to handle more promising results first (Carver and Lesser, 1994). Only the central process managing the blackboard is able to employ such a strategy, while individual components solely rely on the information available locally. On the other hand, this central storage and control schema presents a considerable architectonic burden, since the possibilities for introducing parallel processing as a means to improve the efficiency of a system are greatly reduced (cf. Kesseler, 1994):

- The concurrent access to the blackboard by multiple processes (which are assumed to realize one component each) may lead to a situation in which the bandwidth of the memory bus may not be big enough to handle all requests instantly.

The bus interface has to delay accesses which reduces the theoretically possible throughput.

- Even if the restricted bandwidth of the bus is neglected, access to the blackboard has to be partially serialized, because only one process may write on the blackboard at a time. In the worst case, this may lead to a complete serialization of write accesses to the blackboard.

A partial solution to this problem is given by distributed blackboards (Jagannathan, Dodhiawala and Baum (eds.), 1989, Part II). By using a distributed blackboard the first effect mentioned before, the serialization because of a restricted bandwidth of memory buses, can be avoided. But even using a distributed implementation of a blackboard, the requirement of serialization of write accesses to the central storage remains. Denecke (1997) proposes a distributed blackboard system, which uses a set of expert-system type rules to control the order of component communication. These rules describe when and under what circumstances a component receives information from the blackboard and is thus able to process data. Ultimately, this does not change the role of the central control component, here called *discourse blackboard*. All actions of components are serialized.

Boitet and Seligman (1994) propose an approach they call *whiteboard* which is of some importance for the work described here. Starting from the classic blackboard approach and its sub-optimality they construct an architecture schema that promises to solve some of the problems. They characterize two main obstacles that result from the use of a conventional sequential system: information loss and lack of robustness. To demonstrate both problems they point to an earlier system, *Asura* (Morimoto *et al.*, 1993). In that system, a component does not transmit all information that was processed to its successors. In many cases, this leads to a partial reanalysis of the input. Moreover, partial analyses of the input are not delivered, because they do not represent complete descriptions of all the input (e.g. syntactical analyses that are well-formed but do not cover the whole input). In some cases, this lets the whole system fail, although at least partial translations would have been possible.[3]

Following Boitet and Seligman (1994), the application of a blackboard architecture solves the problem of information loss, but does not contribute to a higher degree of robustness in a system. Additionally, other complications arise, mainly because of the concurrency of components accessing the blackboard. Examples of these are lack of efficiency and complex localization of errors.

The whiteboard approach contributes to the solution of these difficulties, say Boitet and Seligman (1994). The architecture schema is shown in Figure 4.1. A central data structure is managed by a process called coordinator. This coordinator receives results from components and sends data to modules. All information gathered by the coordinator is stored, together with some kind of reason maintenance (Doyle, 1979), which relies on the path a data element has taken through the application. The type of data structure is irrelevant in principle, but the use of a lattice

[3] INTARC (cf. Section 1.3) shows a similar behavior.

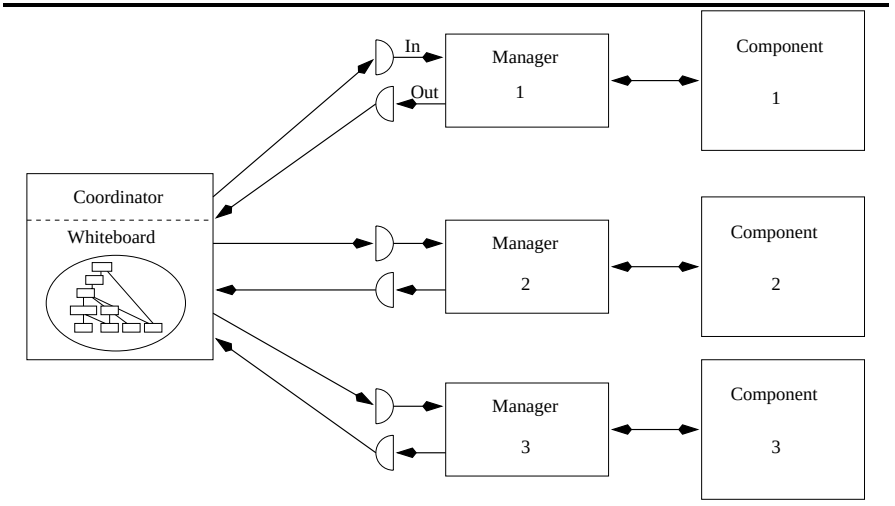


Figure 4.1. The *Whiteboard*-Architecture (from Boitet and Seligman, 1994)

structure which is claimed to be appropriate for the processing of spoken language is proposed. The internal structure of individual components is not important either, because each component is encapsulated by a so-called manager. These managers maintain queues of incoming and outgoing messages and operate on them periodically. This also makes the data format of messages irrelevant for the most part, because managers may initiate conversion processes to adapt message formats to what components expect. This makes the reuse of existing components easier.

Besides the storage of data, the coordinator may also execute functions which resemble the control processes of a blackboard. This means that the coordinator realizes the central control and may influence the processing of components by selecting which messages to deliver first.

Boitet and Seligman (1994) claim that the incremental behavior of an application can be simulated using a whiteboard. Within the prototype system KASUGA, which uses a whiteboard, this has been demonstrated. KASUGA is a system comprised exclusively from non-incremental components. Incrementality is simulated by holding back the results of a component in the manager and by distributing them in a piecemeal fashion to the coordinator. KASUGA consists of:

- A phoneme recognizer that constructs a graph for the input utterance by computing the three best phoneme candidates for a segment,[4]

[4] This means, obviously, that the recognizer creates a phoneme graph with a transcript-independent density of three.

- A syntactic parser that constructs words from phonemes using a context-free grammar, and
- A bilingual lexicon (Japanese-English) that facilitates a word-for-word translation. The system is able to translate 40 elementary phrases (*bunsetsu*).

The concept of a whiteboard represents a considerable step forward when compared to sequential systems. However, a careful analysis shows that whiteboards do not represent an architecture schema that can fulfill all the requirements stated in the introduction. Even if the components defining a system have no knowledge about each other and the processing strategies involved, there is still a central instance, the coordinator, which not only stores all results, but moreover is able to control the execution of an application. This means that there will be unnecessary serialization of actions, even when using a whiteboard.

The most important feature we demanded in the introduction, the use of incrementality, can only be simulated in the approach discussed here. The smallest consequence of simulating incrementality is a loss of efficiency, because the delay of a component is equal to its overall processing time. The method for withholding information in managers and distributing it piece by piece only simulates an incremental output. There is no information on whether a component may be able to handle incremental input. More important, however, is the lack of interaction between modules. Interaction is not possible because the module which would be the receiver of control information from another component has completed its processing before the information can be received. Thus, KASUGA can at best be seen as an early demonstration system, as far as incrementality is concerned.

The statements made so far lead to certain minimum requirements for a data structure that we see fit to be used for the processing of spontaneous speech. Such a data structure should at least have the following properties:

- The schema to store results is distributed and does not rely on a central place for storing information. Thus, bottlenecks that do not stem from the intrinsic function of a system can be avoided.
- Data can be viewed on several levels of abstraction, in order to simplify the selection and monitoring of information.
- The data structure can be equipped with efficient means of communication between modules in order to exchange data objects. These communication means do not create a considerable overhead for communication.
- The data structure should be accessible from all components in a uniform way. Thus, the exchange of linguistic objects is simple and efficient and does not lead to a loss of meaning of the data objects.
- A uniform formalism to store linguistic objects should be employed. The reasons, again, are transparency and efficiency.

Layered charts as a data structure realize these concepts to a great degree. The underlying principle is illustrated in Figure 4.2. Each intermediate and final result that is created by a component denotes a hypothesis over a certain interval of the input speech signal. Thus, the hypergraph resulting from word recognition forms the

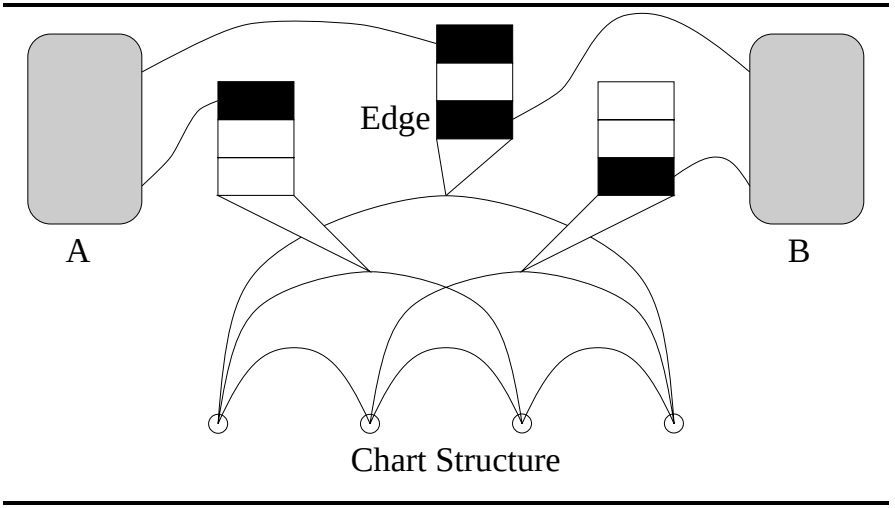


Figure 4.2. The principle layout of a layered chart

lowest level of the layered cart. Building upon that level, other components create additional hypotheses that cover a single word hypothesis to start with. During the course of computation in an application, edges may be combined, e.g. as described in Section 2.6.3. At any give point in time, the distributed layered chart forms an interpretation graph of hypotheses describing the input. The main components of an edge within the chart are:

- **Identification**: A global, system-wide valid identification of an edge is essential for maintaining references that refer to entities outside the local address space of a component.
- **Domain**: The part of the input utterance described by an edge is in practice given through the sets of start and end vertices of that edge. Naturally, the restrictions for hypergraphs have to be maintained.
- **Description**: This is the representation of the linguistic object that is covered by the edge. We use complex, typed feature structures encoded using the formalism described in Chapter 3. The only exception is the initial hypergraph construction by the word recognition component. In that case, orthographic labels representing words suffice.
- **Score**: Each edge is annotated with a score that in principle may have been calculated using several different knowledge sources and evidence. The current version of MILC uses acoustic scores derived from the word recognizer, as well as bigram scores calculated by a stochastic language model.
- **Predecessors**: Each edge is equipped with information regarding reason maintenance. This means that it carries information about which edges have been used

to create it. Word edges generated by the speech recognizer do not have predecessors. Any other edge may have exactly one predecessor, e.g. in the case of lexical access within syntactic processing. Two preceding edges are also possible. This happens when the fundamental rule is applied in one of its variants (see below). As the fundamental rule combines exactly two edges to form a new edge, there are never more than two predecessors to an edge. The complementary relation, the successor information, is also stored for each edge. This simplifies inheritance operations tremendously. Inheritance of information takes place if the word recognition integrates a basic word edge into an already existing hyperedge. In this case, the information about the modified sets of start and end vertices, as well as the possibly augmented acoustic score, have to be transmitted to other components to enable them to update their local information accordingly. If the modified hyperedge has already been used to construct other edges by some linguistic operation, these have to be updated, too.

- **Bookkeeping**: In addition to the fields described so far, an edge carries with it a whole set of administrative information, which either can be computed from the information already described[5] or is not linguistically relevant (e.g. configuration options about how to print an edge etc.).

Layered charts are an extension of charts, as they have been originally proposed by Kay (1973) for the structural analysis of natural language. The originating thought was the following: Several partial analyses can be used many times during an analysis, so it might be helpful to store intermediate results in a so-called *well-formed substring table*. This table may be extended by additionally storing incomplete analyses. This (little) modification renders far-reaching consequences for the complexity of algorithms for natural language processing. In the case of syntactic parsing with context-free grammars, the complexity can be reduced from exponential complexity to $O(n^3)$ (Sheil, 1976).

At the time when charts were introduced, incomplete analyses were always connected to phrase structure rules whose right sides have not yet been fully consumed. Such edges are called *active* in order to hint at the fact that they are still looking for further material before the analysis is complete. *Inactive edges*, on the other hand, represent analyses that have completely carried out and can either be printed as result or wait for the incorporation into a bigger structure.

Definition 4.1.1 (Chart).
A chart is an acyclic, directed, left-connected graph with edge labels. The set of edges is partitioned into two disjunct subsets:

- *Active edges represent incomplete analyses.*
- *Inactive edges represent complete analyses.*

Additionally, a function is defined that decides if an inactive edge should be considered a result and be printed or not.

[5] This information may be stored redundantly for efficiency reasons.

In order to carry out an analysis with the aid of a chart, usually three central procedures are defined: The insertion of edges into the chart (INSERT), the introduction of new hypotheses on the basis of completed analyses (PROPOSE), and the fundamental rule (COMBINE). Here, we will only delve into some aspects of the fundamental rule and point to the standard literature for other aspects of chart processing (Winograd, 1983; Thompson and Ritchie, 1984; Allen, 1987; Görz, 1988).

The fundamental rule combines two edges and constructs a new edge in case of a success. Usually (e.g. in a conventional system for structural parsing), an active edge is combined with an inactive edge. The end vertex of the active edge has to coincide with the start vertex of the inactive edge. The result of the combination is an edge that spans over the two arguments used. However, other combinations are possible as well, e.g. cases in which the inactive edge is on the left side (so-called island analysis, cf. Section 4.7), or combinations of two active edges (this strategy is not used within MILC). It may be reasonable not to make the decision about the combination of two edges based on a common vertex. For example, in the case of a transfer based on semantic descriptions of intervals of an incoming speech signal, the surface order in which the constituents in question appear is not relevant. More important is that all parts of the semantic content of an edge are explained by corresponding partial analyses. This requires that an active transfer edge may incorporate inactive edges that are covered by it, regardless of their exact position.[6]

Another domain in which the combination of edges is not strictly bound to adjacency in time is generation. Kay (1996) describes the surface generation of sentences from shallow semantic descriptions that resemble the minimal recursive terms used in Verbmobil (Copestake *et al.*, 1995; Dorna and Emele, 1996). The goal of generation is to explain the sequence of singular terms with a surface representation. The domain of each edge is the subset of indices into the sequence of semantic terms that have been used for its construction. Thus, two edges may only be combined if the intersection of both domains is empty; the domain of the combination result is the union of both source domains.

To have the data structure of a chart and the three procedures briefly mentioned is not enough to specify an actual component that could fulfill any reasonable linguistic task. To achieve this, two further ingredients have to be defined: An *agenda* and a *processing strategy*.

In this context, an agenda is always a place in memory where tasks are stored until they are carried out. Each of the tasks describes the execution of one of the chart procedures with a specific set of elements. Kay (1980) calls this *algorithm schema*, which guarantees that finally all necessary operations will have been carried out, all tasks will have been executed. However, the schema leaves open the exact order in which tasks are chosen. Only by specifying the exact order of application does the algorithm schema become an algorithm, which defines the exact sequence of analysis. We will describe some possible strategies in Section 4.6. The layered

[6] An earlier version of the approach discussed here (Amtrup, 1994b) was grounded on structural transfer based on syntactic criteria; in that case, a surface-oriented method that pays close attention to the structure of the underlying graph is applicable.

chart provides a global algorithm schema. Each of the components has to specify certain functions defining the exact model for the chart, depending on the actual needs of the analysis.

In MILC, each module has its own agenda. The agenda is created global to the module and is consulted in short intervals. The length of the intervals depends on the respective processing strategy. Usually, each time a new vertex is inserted into the chart, a processing cycle is initiated for the agenda. This corresponds to an incremental strategy that attempts to completely finish the processing of an interval in time before new input for later intervals is considered. That means that all tasks pertaining to earlier points in time should be carried out before any task for a later point in time is processed. However, this rigid schema can be partially broken due to information from the reason maintenance for edges.

The reason for this behavior is that components within MILC always use a beam search (Steinbiß, Tran and Ney, 1994) to process an agenda. This entails that usually not all tasks contained in the agenda are carried out in one processing cycle. Instead, the component defines a boundary score for tasks, which is taken to be the criterion for deciding whether a task is "important" enough to be carried out or not. All tasks above that threshold are considered, all tasks below it are not initially. Several systems get rid of the tasks below the threshold altogether. This is not possible in MILC, as the score of edges (and consequently, of tasks) may be changed. To enable this, the tasks within a component are not only globally stored, but for each edge there is a local agenda that stores the tasks relevant for each edge.

The rescoring of a task is necessary if the acoustic score of an edge changes due to the insertion of new word hypotheses into the hypergraph. The modified acoustic score is given by the word recognizer. We already mentioned that reason maintenance information is part of the data kept for each edge. If the score of a word edge changes, then the modified score has to be inherited from all edges that have been constructed using the word edge. This may have the effect that a task that affects a modified edge and has not been carried out is now just above the threshold that would have justified a processing. All components track this type of situation and initiate the processing of a task, even if the interval in time that is affected by the task has already been completed. Thus, the strict incrementality of the aforementioned agenda strategy may be changed under certain circumstances.

Not only the acoustic score changes a basic word edge and leads to the inheritance of information to other edges; the modification of the set of start and end vertices also gives rise to this schema. However, in this case the agenda mechanism does not have to retract in time.

There is one further important property of a layered chart that has not been mentioned yet: The graph structure, which is the basis of the processing of speech using word graphs, may be kept active for the major part of the application. All implementations of NLP systems that we know of abandon the graph structure at some point during processing. At a certain point, the system makes a decision for a specific reading of the input and usually does not use a graph-like representation afterwards. The prototypes of Verbmobil (Bub, Wahlster and Waibel, 1997) have placed this

border rather early, namely just behind the syntactic parsing stage (Block, 1997). The architecture prototype INTARC (Görz *et al.*, 1996) moved this line in front of transfer. With the aid of layered charts, MILC is able to keep the graph structure alive into the surface generation. As a consequence of this, sequences of surface constructs can be presented as output. The user notices this mechanism only by acknowledging that the generation process is incremental in itself and displays utterance parts that change with time. A similar strategy is proposed by Finkler (1996), whose generator processes the semantic representations of the content to be uttered incrementally. He observes repair phenomena and restarts.

The explanations so far finally let us define a layered chart somewhat formally. However, we neglect all information needed to carry out an analysis.

Definition 4.1.2 (Layered Chart).
A layered chart is a directed, acyclic, left-connected hypergraph[7] *with vertex and edge labels. Each edge contains the following information:*

- *A system-wide unique identification taht is generated as a pair of the creating component and a locally unique identification,*
- *a set of start vertices (resulting from the hypergraph property),*
- *a set of end vertices,*
- *a description (in our case usually a typed complex feature structure),*
- *a score, that may contain several components (e.g. an acoustic score and a language model score),*
- *the identities of the two hyperedges which were used for the construction of this edge, and*
- *a local agenda that is used to store tasks that could not be carried out due to the beam search used in the component.*

4.2 Communication Within the Application

Layered charts represent a reasonable and very useful data structure in the framework of incremental natural language processing. In order to successfully distribute such a complex structure on multiple processes or machines, efficient communication mechanisms are needed, which are implemented according to the immediate needs of the task. On one hand, the infra-structural level should be specified in a sufficiently abstract manner so that the communication primitives that realize the transmission on a low level need not be taken into consideration. On the other hand, it should be simultaneously possible to control options of the communication behavior.

The realization of the infrastructure should be grounded on a solid theoretical foundation, but it should also be as independent as possible of the actual implementation of a module. Under no circumstances is it reasonable to decide anew

[7] Note that this leads to an association between nodes and points in time. Later (See Section 4.8) this will become important.

about the optimal type of implementation for each interface that is developed. To the contrary, the framework of communication must be designed ahead of the implementation of an application. This is the case for every non-trivial AI application, in particular for MILC, since it is a highly complex, distributed system with several components and connections among them. The communication system ICE (*I*ntarc *C*ommunication *E*nvironment, Amtrup, 1994a), which was developed in the framework described here, is an example of such an acceptable subsystem. ICE has been integrated successfully into several projects for natural language processing, among them all prototypes of the Verbmobil system (Amtrup and Benra, 1996), and the architectonic prototype INTARC (Amtrup, 1997a). The use within Viena (Wachsmuth and Cao, 1995) is an example for the integration into a system outside the narrow range of speech translation. Viena is a multi-modal system that enables intelligent communication with a system for virtual reality.

The channel model CSP (*C*ommunicating *S*equential *P*rocesses) forms the theoretical foundation for ICE. CSP (Hoare, 1978) defines the construction and semantics of point to point connections between processes. A successful implementation of the CSP model has been reached with the transputer (Graham and King, 1990) hardware and the corresponding programming language developed for it, Occam (Burns, 1988). Channels are bidirectional connections between Occam processes that can be used to exchange messages. Such a message-oriented kind of connection is best suited for the type of application we are interested in. Using shared memory is not possible for reasons already stated above (bus contention, serialization of write accesses). The other main communication paradigm currently available, remote procedure calls (RPC), seems to be disadvantageous, too. RPC uses a *rendez-vous* synchronization schema that may show unacceptably high connection times due to network latencies. This has also been the reason for modifying the original model of Hoare (1978) for use in ICE. CSP uses a *rendez-vous* synchronization prior to the actual exchange of messages. The communication model used for ICE employs asynchronous message passing, which does not suffer from network latencies at all. The second major divergence from CSP is concerned with the configuration of channels, which results in different kinds of channels.

4.2.1 Communication Architecture of an Application

The facet of the overall architecture of an application that covers communication aspects is shown in Figure 4.3. An application may consist of any number of components. These components could be implemented in one of several programming languages. The languages currently supported are C, C++, Lisp (Allegro, Lucid, CLISP, Harlequin), Prolog (Quintus, Sicstus), Java and Tcl/Tk.[8] MILC, however, does not actively use the possibility of heterogeneity; it is completely implemented using C++.[9]

[8] The interfaces for CLISP and Java have been implemented by Volker Weber.

[9] The only exception from this rule is the visualization component, cf. Section 4.10. However, even there the communication mechanism is implemented in an underlying C-layer.

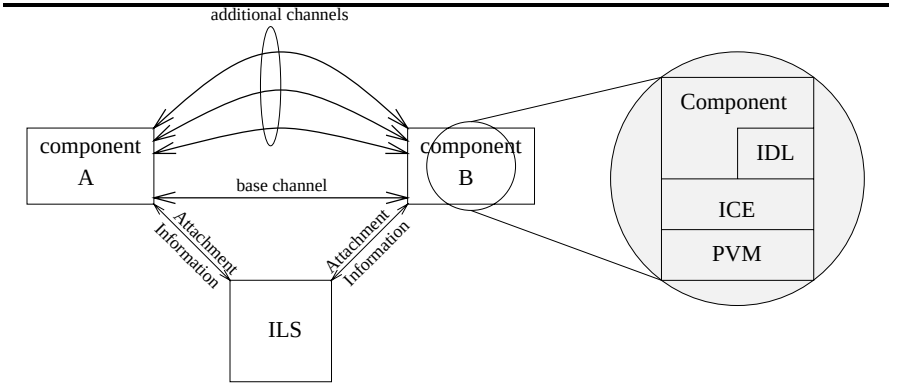


Figure 4.3. Principle component layout

One possible extension of the architectural view on communication could have been to adopt the notion of agents cooperating to a certain degree while carrying out a certain task cooperatively, but this would have meant to mix up different conceptual levels of a system: The communication facilities we are describing here establish the means by which pieces of software may communicate with each other. They do not prescribe the engineering approaches used to implement the individual software components themselves. We do not state that agent architectures (e.g. Cohen *et al.*, 1994) cannot be realized with our mechanism[10], but the range of cases where ICE can be applied is broader than this.

All communication within the systems is carried out via channels, i.e. bidirectional, asynchronous point-to-point connections. We refrained from implementing the underlying primitives from scratch. Instead, we used PVM (*P*arallel *V*irtual *M*achine, Geist *et al.*, 1994), a message passing software whose usage is extremely widespread nowadays. PVM is able to let a heterogeneous network of workstations act as if it were one single large virtual parallel machine. The immediate consequence of adopting this approach for the implementation of ICE is the fact that applications using ICE can be distributed over heterogeneous systems as well. In practice, we have distributed systems on machines from Sun (Solaris), HP (HPUX), IBM (AIX), SGI (Irix) and on standard PCs (Linux). The most important benefit of using a preexisting software layer for the very basic issues is the amount of debugging that has been invested into that layer. Earlier, disillusive experiences with a predecessor system (Pyka, 1992a) made it very clear that a extremely stable basic communication software is essential.

[10] Indeed, distributed blackboards as used in (Cohen *et al.*, 1994) can easily be modeled using a channel-based approach.

On top of the PVM layer the core level of the communication software is situated (ICE in Figure 4.3). This level contains the functions needed for attachment and detachment of components, sending and receiving of messages, and a number of administrative functions governing the general behavior of components in a system. Moreover, this level contains the interfaces to the different programming languages that can be used in an application. The topmost, optional, layer of functions (IDL, *I*ntarc *D*ata *L*ayer) handles the communication of complex data types. As we already mentioned, user-defined data objects can be transmitted transparently using ICE. In order to do this, the only code that has to be supplied are encoding and decoding routines for the desired data type. After registering the data type within ICE there is no difference between exchanging basic data types and complex ones.[11] The primary use of this layer of ICE contains the definition of chart edges that are exchanged within the application.

ICE defines one specialized component, the ILS (*I*ntarc *L*icense *S*erver) that operates as configuration center for an application. Each component that takes part in a system attaches to the application using the ILS; detachment is handled similarly. Additionally, the ILS keeps track of the configuration of all channels being used during the runtime of a system. However, the ILS does not constitute a central data storage and does not have central control rights. After the establishment of a channel between two components, the communication is carried out strictly between those components; it is therefore strongly decentralized.

4.2.2 Channel Models

The interchange of messages among components takes place using channels. We distinguish two different types of channels: *Base channels* and *additional channels*. Base channels are the primary facilities for communication. They are configured to guarantee that each component is able to interact with any other component it wishes to, regardless of programming languages, hardware architectures, or system software being used. This is achieved by using the standard communication mode of PVM, which supports XDR[12]. Message passing is done asynchronously. There is no need for a configuration negotiation or localization of modules. Additional channels, on the other hand, can be configured in various ways, e.g. not to use the hardware-independent encoding schema XDR in order to improve communication performance. Moreover, using additional channels, the data stream between two components can be separated conceptually, e.g. to divide data and control streams.

[11] Originally, the design and implementation of an abstract language had been planned defining a common set of primitive data types which would have been accessible from all programming languages (Pyka, 1992b). Departing from that language, the implementation of compilers was planned that would have generated the necessary functions in different programming languages. However, this approach turned out to be too restrictive, the only interfaces fully implemented being the ones for C and C++.

[12] e*X*ternal *D*ata *R*epresentation, see Corbin (1990), an encoding schema for data objects independent of the current programming environment.

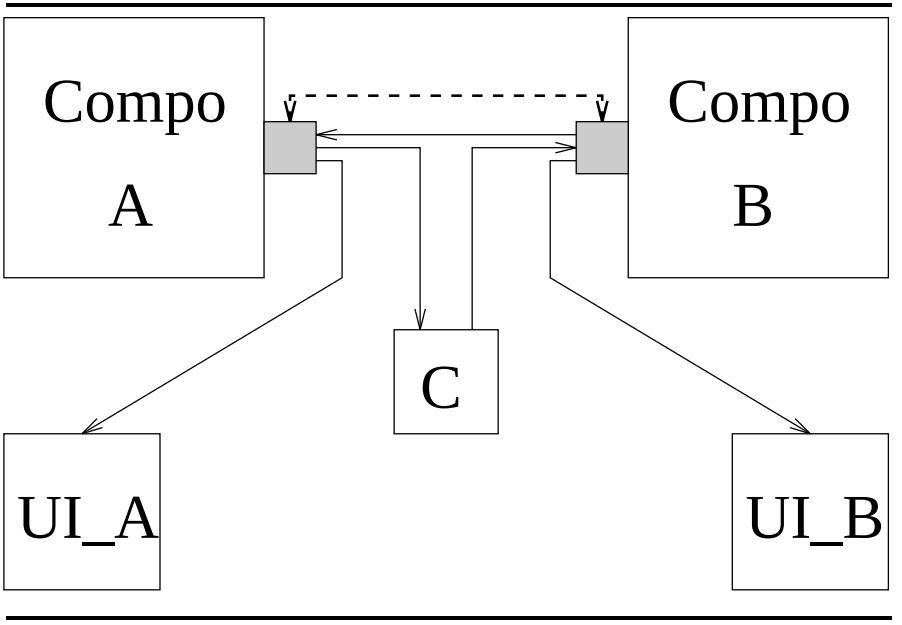


Figure 4.4. The configuration of *split channels*

Both types of channels are prepared to transmit both basic and user-defined data types. This property is similar to Occam channels, which can also be equipped with a channel protocol. Interestingly, the definition of this kind of data types (records) has not been carried over into the language definition. The effect is that complex structures may be transmitted, but they cannot be used in the receiving process without some difficulty and programming effort.

To allow for further flexibility in system design, both base and additional channels can be configured in a way that modifies the topology of the application. That way, they do not only work as data pipelines between exactly two components. Instead, the application can define a number of components as listeners on certain channels. In the same way it is possible to define multiple potential message sources for a channel. The motives for the definition of *split channels* is mainly twofold:

- The realization of visualization components for data that are sent along a channel. That way, not only the search for programming bugs can be simplified considerably, but moreover there is the possibility of displaying partial results to the user in an easy way.
- The adaptation of different data formats between two components. Inserting an additional module that is designed to modify data is especially helpful if version

changes for parts of the application lead to changes in the interface definition between components.[13]

Consider Figure 4.4 as an example for a configuration using split channels. Two components, **A** and **B**, are connected using a channel that is depicted by a dashed line. The channel endpoints (grey boxes in the figure) are split up to allow visualization of message data sent by either component. The visualization is performed by two additional components labeled **UI_A** and **UI_B**. Furthermore, the data sent by component **A** must undergo some modification while being transported to component **B**. Thus, another component **C** has been configured that is capable of transforming the data. It is spliced into the data path between **A** and **B**. Note that data sent by component **B** arrives at **A** unaffected from modification by component **C**.

The configuration of split channels is completely transparent for the components involved. The behavior of an application thus does not change simply by the topological change resulting from the configuration of a split channel. Of course, the components affected by the splitting can alter the behavior of the system. Using this kind of configuration, the breadboard architecture of components for Verbmobil has been developed (Bub, Wahlster and Waibel, 1997).

4.2.3 Information Service and Synchronization

The ILS takes care of three functions highly relevant for the execution of a distributed application:

- The storage and administration of names of components within a system,
- the configuration of the overall application, and
- the initial settings for broadcasting messages to all other components in a system.

Usually, only the ILS has a complete picture of the structure of an application, while individual components only have knowledge of the part important to them. In order to do so, the ILS stores a table of names and locations of modules.

As a consequence, registration at the ILS is one essential part of the initial startup sequence of a module. The attachment is stored in the ILS and the identity of the new component is handed to other modules that notified the ILS earlier of a need to communicate with that component. The amount of synchronization necessary for this startup has been kept as low as possible (see below). Similarly, a component should detach from the application prior to termination. But even if a component terminates due to a program or system failure, the ILS is notified of this event (via PVM). Thus, components may react to the detachment of another module; they are given a notice by the ILS.

The second main function the ILS performs is the management of the channel configuration. It reads a configuration file on startup that describes the layout of split channels. For each channel, a list of real connections is defined. Each real

[13] Within MILC, there was no need for such a component due to the integration of the system. However, this facility has been beneficial in other contexts.

```
A B BASE
UIG_A BASE -1 1 0
C BASE -1 1 0
B BASE -1 0 1

B A BASE
UIG_B BASE -1 1 0
A BASE -1 1 0
C BASE -1 0 1
```

Figure 4.5. An example of a configuration file for split channels

connection is given a direction. The syntax for channel configuration files is defined in Amtrup (1995b). The configuration file necessary to result in the behavior shown in the example earlier given is outlined in Figure 4.5.

During the execution of an application, usually more channels are created by request from components. The only restriction on this is that split channels cannot be dynamically administrated. The configuration for split channels is read once during system initialization.

The third function of the ILS is to provide initial information about the components enrolled in an application to a module that wants to broadcast a message to all other components. During conventional execution of an application, each module only has knowledge about components it directly interacts with. Thus, there might be components unknown to a specific module. However, information about all modules participating in a system is necessary if a message has to be distributed to all components. A centralized approach (resembling a blackboard model) would require sending a broadcast message to a central component (here: the ILS) that takes care of the distribution to all other components. Deviating from this, ICE uses the connection to the ILS only to initialize the broadcasting function for the first time. Again, this results in a low synchronization overhead and prevents a communication bottleneck at the ILS.

The distribution of broadcast messages is done in three steps:

- If a component wants to send a broadcast message for the first time during the runtime of a system, it sends a request to the ILS in order to get information about the identity of all other components. The ILS uses its configuration record about the application and sends the appropriate answers to the inquiring component.
- After the broadcasting component has received the configuration information about all components, the message can be sent on the individual channels to the receivers.
- In the future, the ILS expects the component that issued one broadcast message to do the same again. Therefore, each time a new component enrolls into the system,

the information about that component is given to the broadcaster, regardless of whether or not the request for opening a channel between the two modules was issued. Thus, the communication between a broadcaster and the ILS to receive the configuration information about all components has to be carried out only once.

This kind of configuration is completely transparent for the broadcasting component. Usually, it does not even know how many components exist in a system. All this information is hidden in the ICE layer and thus does not pose a burden for the application developer.

The most interesting part of the communication between components and the ILS is, without doubt, the initial synchronization phase that defines the identity of modules and all necessary channels. One of the design goals for the implementation of ICE was to minimize the synchronization overhead here as well. The reasons for this were first of all to prevent the presence of a communication bottleneck. Second, a single process may contain any number of components, and a lengthy initialization with the ILS may lead to severe performance degradation.

Consequently, a component should only request information about another component if absolutely necessary, i.e. if a message shall be sent to this other component. ICE implements an information profile that is able to handle under-specified component and configuration data. Figure 4.6 depicts the sequence of actions taking place from the attachment of a component up to the first exchange of a message between two components. The outer left and right parts of the figure show the components participating in the application, the center reflects events within the ILS.

Initially, component **A** registers with the ILS and announces its identity. To achieve that, the component calls the function `ICE_Attach()` (or one of the variants in one of the supported programming languages), thereby sending a message with the tag **ILS_ADD** to the ILS.[14] However, **A** does not wait for an acknowledgment, but continues its operation.

After the attachment has been accomplished, **A** requests the establishment of a channel by calling `Ice_AddChan()`. A corresponding message with the tag **ILS_CHC** is sent to the ILS, which answers by sending the required configuration data. These answers contain the real channels used for sending messages with the tag **ILS_CHS**.[15] In this case, there are two instances of real channels on which data is sent. Configuration messages with the tag **ILS_CHR** define real channels used

[14] Besides being typed, messages are also equipped with a tag in order to signal the purpose of a message. The receiving functions of ICE can be parameterized to only look for messages with certain tags. **ILS_ADD** is a predefined tag reserved for system configuration purposes. The exchange of such system messages is done in a way that is transparent to components. During the call of ICE functions, system messages are always received with high priority.

[15] In the case of the configuration of split channels, this may be more than one channel. We use the term *real channel* to hint at the fact that messages are actually sent along these channels, while the notion of *channel* is a mere suitable abstraction.

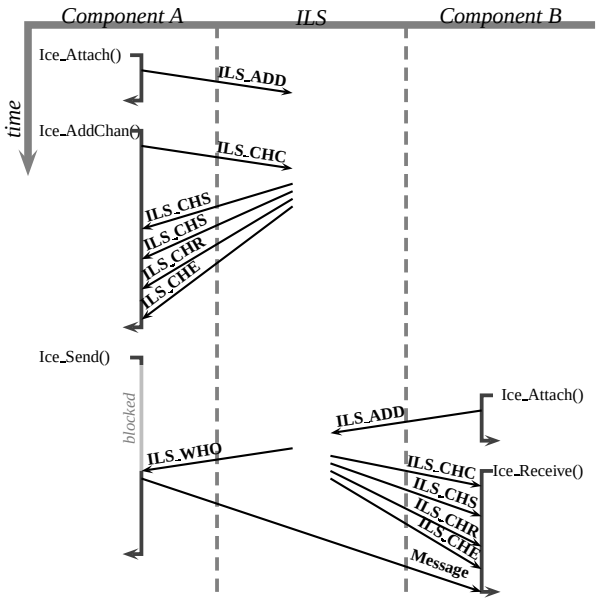


Figure 4.6. Time line of the initial configuration of channels with the ILS

to receive data items. Finally, the completion of the configuration is signaled by sending a message with the tag **ILS_CHE**.

After this configuration has happened, **A** tries to send a message to **B** using the recently established channel. Since the target component **B** is not yet enrolled in the application, the function call is blocked.[16] After some time, the component **B** registers with the ILS. As soon as the ILS possesses this knowledge, it informs the component **A** of the attachment of the target component. Thus, **A** is able to send the message and continues with its operation. Meanwhile, the ILS sends configuration information about the channel to **B**, even though **B** has not requested the establishment of that channel. As soon as **B** calls a receiving function of ICE (in this case `Ice_Receive()`), the configuration data is processed first. The message has meanwhile arrived at **B** and can be processed in a normal way.[17]

[16] This behavior can be changed. The sending component may decide not to wait if one of the target components is not present. Instead, the function call may return and raise an error situation.

[17] The message from **A** could in theory arrive at **B** before the configuration message is present. In this case, it will be buffered, until **B** is completely configured.

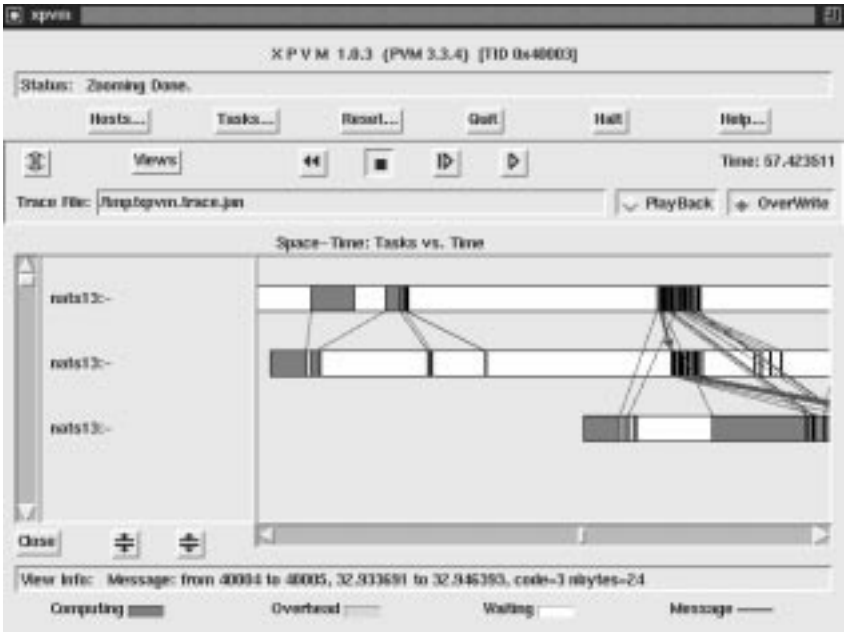


Figure 4.7. XPVM snapshot of the initial synchronization

Figure 4.7 shows a section of the actual communication during initialization of a system. The snapshot was captured using XPVM, a graphical tool for PVM. The upper bar shows the activity of the ILS, the bar in the middle represents **A** in our example, and the lower bar depicts **B**. The message marked by an arrow is the one notifying **A** that **B** arrived.

4.2.4 Termination

The termination of an application is not in itself part of the definition of ICE. However, since the communication status is closely related to termination, we will discuss termination in MILC briefly in this section. MILC is not designed to be a continuously running system, but terminates after having processed one utterance (but cf. Section 4.11). Therefore, it needs to recognize this point in time.[18] Due to possible feedback loops in the topology of an application, it is not possible to assume a unidirectional information flow that would enable a termination schema

[18] This is a necessity even for continuously running systems. The only difference is that in that case the application is not terminated and only a few data structures have to be reset.

that simply terminates components as they have consumed their primary input. In this case, only the first component (the word recognizer) would have to receive a signal that marks the end of the input (in our application this signal is given directly by the structure of the input, cf. Section 4.4). In the current version, the graph of components is in fact acyclic (cf. the following section). However, in our view the current functionality should only be viewed as a starting point for further research and experimentation into the architecture of natural language systems. Thus, we implemented a distributed termination for MILC.

The termination protocol (cf. Mattern, 1987; Peres and Rozoy, 1991) uses message counters. These counters record how many messages have been exchanged between any two components. Consequently, each component keeps two counters per connection. The system may terminate if the components pairwise agree about the number of messages sent and received, i.e. if every message that has been sent was also received by the addressee. This consistence condition could be checked periodically. In the case of MILC, however, it is reasonable to initiate the examination of the termination condition only in case the word recognizer, which is the root of the application graph, has finished its processing.

4.3 Overview of the Architecture of MILC

Following the introduction into the concepts behind layered charts and the description of the software layer responsible for communication between components, we can now start to give details on the application architecture of MILC (Amtrup, 1998). MILC is the first instance of a system based on layered charts. Some interesting aspects of architecture have so far been neglected in the current design; however, the architectural framework allows the integration of new methods and the addition of other components without any major problems. A rough architecture sketch of MILC is given in Figure 1.2, which we repeat here as Figure 4.8.

As previously said, the graph-like structure in the middle of the figure symbolizes the use of a layered chart by the modules of the system. This means that there is always a consistent overall state of the system, which can be retrieved by taking the union of all edges present in the system. In order to facilitate debugging of the application, and for demonstration purposes, we developed a visualization component that is able to graphically display the edges present in a layered chart. This visualization is described in Section 4.10, but is omitted in Figure 4.8. The arrows symbolize ICE channels, the heads point in the direction of the information flow.

The root component of the application is given by the **word recognition**. We use word graphs that have been created incrementally by the Hamburg speech recognizer (Huebener, Jost and Heine, 1996), thus the graphs are left-connected. The actual recognition process has not been integrated into the runtime system of MILC, as the recognition in itself is a highly time-consuming task. In that sense, the name “word recognizer” is not completely accurate. The function that the component carries out is to read a word graph and to convert it incrementally into a hypergraph.

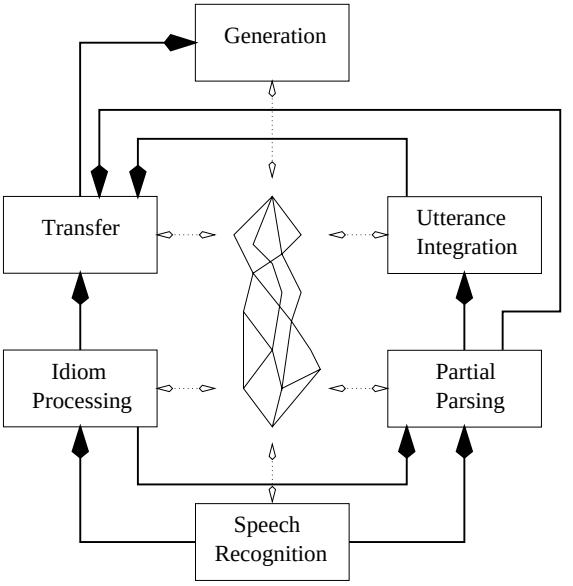


Figure 4.8. The architectonic overview of MILC

Three components are present for the syntactic-semantic analysis of input utterances: **Idiom recognition**, **partial parsing** and **utterance integration**. We will describe them in detail in the following sections.

The **transfer** component maps the semantic descriptions of German utterances into corresponding English semantic representations. Even this phase is chart-driven, however, it is not grounded on syntactic criteria as a preceding study (Amtrup, 1995a) did.

Finally, the English semantic representation is subject to a **surface generation** process, the output of which consists of a sequence of potential English utterance fragments that represent the current best path through the generation graph.

4.4 Word Recognition

The component responsible for word recognition takes as input a left-connected word graph and uses it to incrementally create a left-connected hypergraph of word hypotheses. The hyperedges are incrementally delivered to the idiom recognition and the partial parsing components.

```
%TURN: n002k000
BEGIN_LATTICE
...
1  38 <sil>  -2239.464600 1  38
1  42 <sil>  -2570.498779 1  42
1  46 <sil>  -2914.757568 1  46
1  52 <sil>  -3504.763428 1  52
38 52 okay   -1290.138916 38 52
42 52 sch"on -988.619324  42 52
38 53 okay   -1354.200317 38 53
42 53 sch"on -1059.124756 42 53
46 53 wir    -718.457947  46 53
38 54 okay   -1430.380493 38 54
42 54 sch"on -1127.405151 42 54
46 54 wir    -791.802673  46 54
40 54 Herr   -1312.388672 40 54
38 55 okay   -1516.757080 38 55
42 55 sch"on -1181.626709 42 55
46 55 wir    -874.704224  46 55
42 55 schon  -1215.008667 42 55
...
END_LATTICE
```

Figure 4.9. A word graph in Verbmobil syntax

Let us take the word graph in Figure 4.9 as an example graph and an illustration of the data format used. Each word hypothesis consists of abstract start and end vertices, the orthographic transcription of the word that has been recognized, an acoustic score (the logarithm of a density that gives a measure of how well the model for that particular word matches with the actual input signal; a small absolute value of the score is considered a good match), and information about the time interval spanned by the hypothesis (start and end point measured in frames of 10ms). Word hypotheses are sorted according to their end vertex and their start vertex.

The task of the word recognizer for MILC is to read that kind of graph and to create from it the lowest level of a layered chart. The algorithm used for this purpose has already been defined as Algorithm 12 in Section 2.6.2. The recognizer starts with an initially empty word hypergraph and successively adds word hypotheses from the input graph. There are two points during the insertion which require communication with the idiom recognition and the partial parser:

- Every time a new hyperedge is created (line [5] in Algorithm 12), this fact is communicated.
- Every time the representation of a hyperedge changes (line [2] in Algorithm 12), these changes are also sent to the receiving components. The possible changes

Table 4.1. Messages sent from the word recognizer to the successor components

No.	Type	Content	Affected hyperedge
...			
1	New hyperedge	$\mathcal{H}1$:<1–38(sil),2239>[19]	$\mathcal{H}1$
2	New end vertex	42	$\mathcal{H}1$
3	New end vertex	46	$\mathcal{H}1$
4	New end vertex	52	$\mathcal{H}1$
5	New hyperedge	$\mathcal{H}2$:<38–52(okay),1290>	$\mathcal{H}2$
6	New hyperedge	$\mathcal{H}3$:<42–52(schön),988>	$\mathcal{H}3$
7	New end vertex	53	$\mathcal{H}2$
	New acoustic score	1354	$\mathcal{H}2$
8	New end vertex	53	$\mathcal{H}3$
	New acoustic score	1059	$\mathcal{H}3$
...			

are the addition of a vertex to the set of start or end vertices of an edge, and the modification (i.e. augmentation) of the acoustic score of a hyperedge (cf. Weber, 1995, p. 78).

Consequently, the processing of the graph shown in Figure 4.9 results in a stream of messages. These messages are schematically depicted in Table 4.1.

4.5 Idiom Processing

Idiomatic constructions like "tut mir leid" ("I am sorry") are usually hard to analyze compositionally.[20] Therefore, the goal is to treat those almost lexically and provide an analysis for the complete phrase. The approaches conventionally used are either to allow lexical entries that contain more than one word as citation form, or to formulate syntactic rules that have specialized subcategorization frames for an idiomatic phrase. The main drawback of both methods is that the method for analyzing idioms is not specialized, i.e. fixed phrases are treated in the same manner as other syntactic constructions. In particular, there are no means for preventing a component from attempting to integrate the individual parts of an idiom into an analysis,

[19] The acoustic score given here (2239) reflects the total score for the interval in time covered by the edge. The criterion for comparison of scores, however, is a measure normalized per frame.

[20] See Schöllhammer (1997) and the literature cited therein for a classification of phraseological terms.

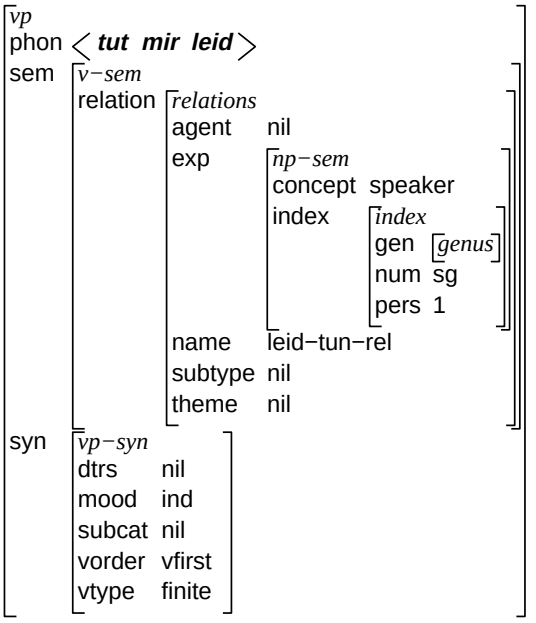


Figure 4.10. Idiom definition for the partial utterance "tut mir leid" ("I am sorry")

even though the character of idioms is inherently non-compositional. Thus, the solution we take for MILC is to implement a specialized component for the recognition and processing of idioms, which aids in reducing the overhead just described.

Idioms seem to play an important role for the domain of appointment scheduling. Their frequency is quite high (Schöllhammer (1997) reports that more than 30% of all utterances contain idioms) and they strongly support the realization of communicative goals by constituting dialog acts (cf. Jekat *et al.* (1995) for an account of dialog acts).

By treating idioms separately from other linguistic phenomena, a fast incremental search can be implemented that only uses the orthographic surface form of words and does not carry out any complex linguistic operations. In order to achieve this, a graph of recognized prefixes of potential idioms is created in parallel to the hypergraph of word hypotheses. In practice, this is done by initially reading a set of idiom definitions of the kind shown in Figure 4.10. The value of the phon feature consists of a list of words that define the idiom. According to these words, a search tree is constructed with edges that are labeled with words being part of idioms. The nodes of the tree are annotated with feature structures if a path from the root of the tree to the node constitutes an idiom.

During the operation of the idiom processing module, the system constantly tries to compute an intersection between the definition tree and partial graphs from the word recognition. In order to do this, tokens are placed on vertices of the hypergraph that mark if and which part of an idiom ends at that vertex. Using this information, it is possible to decide whether an idiomatic phrase has been completed, in which case a corresponding message is distributed to the successor components.

The information about an idiom consists of two parts. First, the feature structure representing the idiom has to be delivered to partial parsing and transfer, of course. This enables these components to incorporate the idiom correctly into their analyses. Second, and equally important, the idiom recognizer distributes inhibition messages. This is done in order to prevent the conventional, compositional analysis of the parts of the idiom by reducing their probability. The background assumption justifying this measure is that most probably the word hypotheses being part of the idiom only have a very limited value for a standard analysis.

However, it is important to notice that the affected word hypotheses are not simply deleted and removed from the hypergraph. This procedure could indeed lead to errors, e.g. in the case where the sequence of word hypotheses mistakenly taken to be an idiom contains words from two correct constituents within a deep analysis. Therefore, only the score of words that are part of an idiom are modified. They receive a penalty which affects the probability of being integrated into a different syntactic context. The method by which inhibition messages are distributed and inherited has already been described.

4.6 Parsing

Usually, most modern systems for natural language processing carry out a complete syntactic-semantic analysis in order to process input utterances. Approaches that use partial parsing[21] are found almost exclusively in the framework of information extraction, e.g. in the MUC-systems (*Message Understanding Conference*). Moreover, the second phase of the Verbmobil project promises to use the partial parsing paradigm to a greater degree (Light, 1996; Worm and Rupp, 1998; Worm, 1998). Partial parsing is also used to reevaluate recognition scores with the aid of licensing analyses of a chunk parser (Zechner and Waibel, 1998). We will also divide the monolithic task of constructing a grammatical structure for input utterances. During the first phase of the analysis, a set of partial analyses shall be found, guided by a grammar mostly incorporating syntactic knowledge, e.g. data for nominal phrases, prepositional phrases, adverbial phrases, etc. Left out will be the construction of subcategorization frames that are common in the description of verbs. Also, we

[21] By partial parsing we mean that an input utterance at hand is not incorporated into one single complete analysis. Instead, a set of analyses is constructed, the elements of which each describe a specific interval in time of the input. This approach is sometimes also called *chunk parsing* (Abney, 1991; Abney, 1996).

will not resolve attachment problems that result e.g. from the ambiguity in the attachment of prepositional phrases. Those derivation steps are taken care of in the second phase of the analysis, which works with the results of the first step. This following phase uses more detailed information in order to be able to model complex verb phrases.

The motivation to use such a two-stage method stems mostly from experiences that we gathered during an experiment with a spontaneously spoken dialog (Dialog N002K from the Verbmobil corpus). We constructed the necessary linguistic knowledge sources required for a conventional analysis based on sentence constituents. Using the already described formalism, we created a type hierarchy with 345 types that, for the most part, described syntactic phenomena. The semantic parts of the hierarchy were restricted to the attribution of concepts in order to decide selectional restrictions in certain cases. The lexicon contained 413 readings for word forms, the grammar had 80 rules. One part of the grammar was tailored to cover date and time expressions that arise frequently in appointment scheduling dialogs.

The knowledge sources just mentioned have been created using the transliterations of the dialog N002K as guideline. However, with the exception of the lexicon which only covers the words occurring in the dialog, the knowledge sources are more general. After the development of the grammars was completed, we carried out experiments using incrementally constructed word graphs, using the standard configuration of the parser. Of course, this specific experimental setting does not allow us to draw general conclusions, because the material used in the experiments had also been used for developing the grammars and for training the recognizer. Nevertheless, we can show that an approach using complete parsing of utterances is not adequate for the analysis of spontaneously spoken language under the conditions we have described.

There are principle arguments that favor a cascaded, partial approach against the complete analysis of utterances. The following two sections discuss the unnecessary construction of verbal complexes, and some properties of spoken language and word graphs that let the approach taken here seem reasonable.

4.6.1 Derivation of Verbal Complexes

The conventional style of generating verb phrases based on the subcategorization information of the verb usually fails in incremental speech parsers. This “conventional” style is to equip each verb with a list of sub-categorized constituents (the complements) and to provide syntactic rules that saturate one complement after the other. This works well as long as verb-first or -second positions are encountered. If a verb-last position, like in German subordinate sentences, is processed, the difficulty arises that the subcategorization information arrives after the sub-categorized elements. Apart from the construction of the sub-categorized constituents, the integration into the subcategorization frame has to take place from the right. This can be accomplished using two mechanisms: Either the parser constructs clusters of possible complements that are unified with the subcategorization list in a single analysis

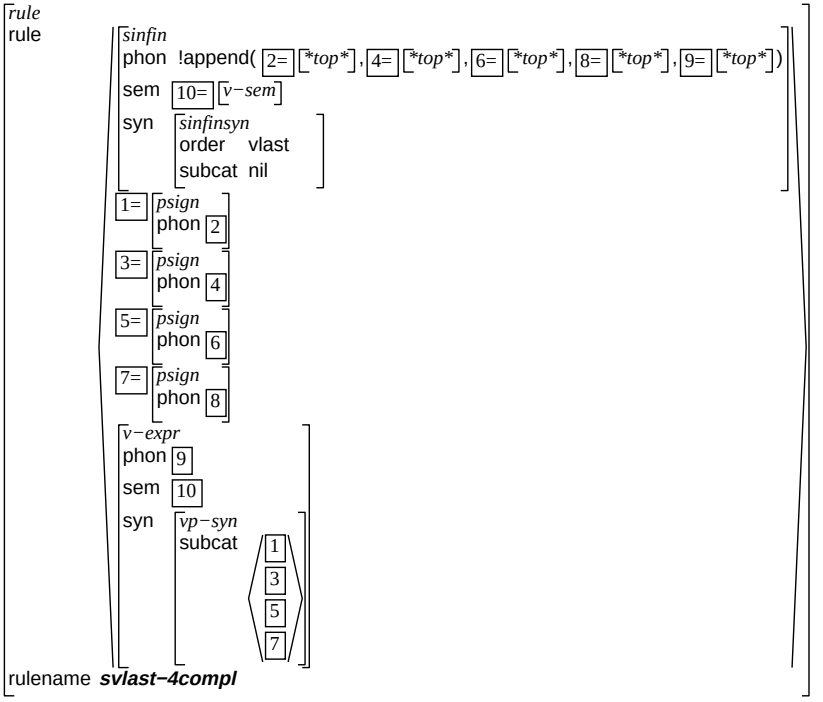


Figure 4.11. A syntactic rule for verbs in last position with four complements

step, or the component provides special algorithms to unify with the subcat list from the right.

The simplest possibility is to provide rules for each possible number of complements that define how to bind the complements for a verb. However, this approach yields a tremendous increase in processing effort. During an experiment, we analyzed a word graph using a preexisting grammar. There were 11 analyses for the turn N002K000.[22] The processing resulted in a chart with 1,362 edges (totaling 4.8 MB) and took 7.07 seconds. Adding one single rule to describe verb phrases with four complements using verb last position (Figure 4.11), the memory consumption raised by a factor of 3.5 (4,655 edges, 19 MB), and the time for completion was 5 times longer (35.9 seconds).

The main factor for this degradation of performance is the construction of clusters of potential complements wherever this is possible. This approach may be acceptable when analyzing written input, but if spoken input is considered, the processing load grows to such an extent that this method is no longer reasonable.

A possible solution for this kind of problem would be to deviate from the strict left-to-right paradigm of analysis. Each grammar rule could be equipped with the information about the "head" of the rule[23], which could trigger the introduction of new edges in the bottom-up step of analysis. The parsing algorithm waits until a matching head is found before applying a grammar rule. After the application, missing elements to the left and right of the head are looked for. This type of head-driven island parsing avoids the unnecessary creation of structures. However, we favor an approach that completely neglects the construction of complement complexes in the first analysis step. The binding with subcategorization lists is left to a successor component, the integrator (see Section 4.7).

4.6.2 Spontaneous Speech and Word Recognition

We have already mentioned elsewhere that spoken language, and spontaneously spoken language in particular, often crosses the border of what would be called a standard grammar for written language (Batliner, Burger and Kiessling, 1994). Frequent phenomena include break-offs of sentences and new starts, but also unusual combinations of syntactic structures. The design of a knowledge source for syntactic analysis must be chosen so as not to reject those structures, but to incorporate even malformed (malformed in the sense of a standard grammar) utterances. If a holistic approach is used for this purpose, an approach that uses a single grammar to model the structure of an utterance, then the combinatorial explosion of the possible analyses may seriously affect the performance of the system. Therefore, cascaded systems are often used for this kind of processing. One example for a cascaded syntactic analysis is given by INTARC (cf. Weber, Amtrup and Spilker, 1997) that extracts the context-free backbone from a large unification-based grammar for a fast stochastic search for possible well-formed candidates (Diagne, Kasper and Krieger, 1995; Kasper *et al.*, 1996).

Besides such a horizontal division of knowledge sources, one could also come up with a vertical division that employs several components for certain parts of an utterance. For instance, Light (1996) presents a system of cascaded finite state automata that try to combine utterance fragments into ever-growing parts. MILC combines both approaches. A vertical division is achieved in which only subunits of the input are processed by the first parsing stage. A horizontal division is carried out by concentrating on syntactic criteria during the first stage of syntactic analysis, while the construction of functor-argument structures, which describe an utterance semantically, is left to the utterance integration.

Another aspect of spoken language, which is reflected in the structure of a word graph, is the occurrence of hesitations and pauses within a continuous utterance.

[22] N002K000 is a turn with multiple segments: Schön hervorragend dann lassen Sie uns doch noch einen Termin ausmachen wann wäre es Ihnen denn recht. (Fine okay then let us schedule another appointment when would it suit you.). One analysis described the first part, 10 the second part.

[23] Not meant exclusively in the linguistic sense of the word.

These phenomena surface in discontinuous chains of word hypotheses, which are difficult to explain and describe with an integrated syntax-semantic approach. At least three remedies are possible, the last two of which we find favorable:

- To formulate rules that allow for hesitations in all possible places,
- to specify that specific syntactic categories may be skipped during an analysis, and
- to modify the parsing algorithms in order to allow gaps between word hypotheses, thereby giving up the strong adjacency demand.

The possibility to ignore edges marked with specific categories is extremely promising. It not only allows a general statement about hesitations or pauses, but can also be used to skip certain words, depending on the context, that might not carry a relevant amount of semantic content. It may be favorable for both methods to divide the modules for syntactic/semantic analysis according to the degree of granularity, either to skip different categories each time, or to treat gaps in the word graph differently depending on the size of the partial analysis. In the framework of MILC, we experimented with the set of categories to be ignored, particularly to be able to ignore certain pertinent particles like in example 4.6.2.[24]

Wir	*können*	*uns*	*ja*	*vielleicht*	*im*	*März*	*treffen*
We	can	us	yes	maybe	in	March	meet

"We could meet in March, probably."

(4.1)

The impact of introducing a partial parsing schema becomes apparent if the processing effort in measures of time and space is evaluated. Table 4.2 presents some numbers on different parameters for the approaches already mentioned. The methods are the complete analysis of transcripts (T), word graphs (NIG) and left-connected word graphs (IG), and the partial analysis of left-connected word graphs (PP). We used the knowledge sources mentioned above, which for example did not include the modeling of discontinuous constituents.

The transition from transcripts to word graphs results in a runtime that is one order of magnitude larger using the same grammar. Using left-connected word graphs further degrades performance. Performing a partial analysis, however, reduced the runtime again to a moderate level. In order to do this, the grammar was stripped of all rules that create complement complexes for verbal phrases and the rules responsible for the attachment of prepositional phrases.

[24] It is at least questionable, though, whether such particles can be ignored or not. In part, they have a great influence on the semantic and pragmatic content of utterances, especially in combination with prosodic phenomena, cf. Niemann *et al.* (1997).

[25] The perplexity of a graph is defined as $p = 2^{\frac{\sum(\log_2(\#_{out}(v)))}{|\mathcal{V}|}}$.

Table 4.2. Results for standard and partial parsing of the dialog n002k

Parameter	Measure	T	NIG	IG	PP
Density (s. Definition 2.3.1)		1	22	181	181
Perplexity[25]		1	3,4	9,7	9,7
Agenda tasks	#	2063	16238	86922	8591
Chart vertices	#	8	50	484	484
Chart edges	#	654	2418	4982	909
Analyses	#	4	4	371	160
Unifications	#	831	3713	13848	1143
Runtime	s	9.37	94.94	113.64	8.56

4.6.3 Structure and Processing Strategies

The input for the module for partial parsing consists of word hypotheses, which are delivered by the word recognizer, and information about idiomatic expressions from the idiom processor. The following types of messages have to be treated:

- **Hyperedges**, which represent sets of word hypotheses of the original recognizer output. They are represented by hyperedges within the layered chart that carry orthographic word forms as labels. The parser carries out a lexical access that may result in several preterminal edges. For each individual lexical reading one hyperedge is created, the structure of which is identical to the originating word edge. The edge label, however, from now on consists of a feature structure that was extracted from the lexicon. For each such preterminal edge, a task is generated to insert the edge into the chart.
- **Modifications of hyperedges**, which happen to surface because of the incremental property of the hypergraph conversion algorithm. We have to distinguish two cases: If the modification is to announce a new start or end vertex to the hyperedge, it has to be examined if new partial paths through the chart are created. In order to do this, the fundamental rule is carried out at the new vertex for all possible combinations of edges. This procedure is only necessary if the affected edge has already been inserted into the chart. If this has not yet happened, the new vertex is processed together with all other vertices. If the modification is to announce the modification (i.e. augmentation) of the acoustic score of an edge, the system has to find out if tasks that have not been considered earlier, because of the threshold during beam search, are now above the threshold. All these tasks are activated.
  In any event, the new information about the edge is inherited by all the other edges that have been created using the modified edge. Also, if necessary, this information is related to other components.

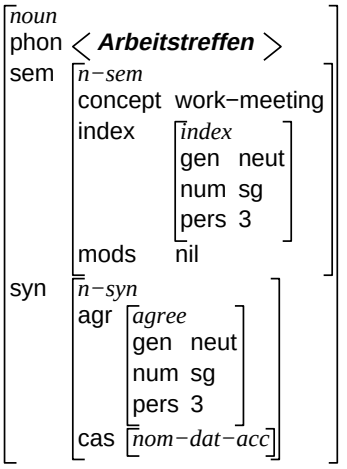


Figure 4.12. One of the lexical entries for "Arbeitstreffen" ("work meeting")

- **Idiom definitions**, which are treated in much the same way as word hypotheses. The only difference is the lack of a lexicon access, because the idiom recognizer already delivers a feature structure that describes all properties of the idiom.
- **Inhibitions**, which affect word hypotheses that were integrated into an idiom. The processing involved here is to attach an additional negative score to the edge, in order to render combinations with other edges less likely. Analyses that have already been created using the now inhibited edge are not revoked, but their score is updated as well. Like the modification of acoustic scores, the presence of inhibition scores is inherited by other edges that have been created using this edge.

Figure 4.12 shows an example of a lexicon entry for the German noun "Arbeitstreffen" ("work meeting"). The entry contains syntactic properties (agreement etc.), as well as semantic information. The semantic concepts are hierarchically organized by specifying them as types in the formalism. The super-types of *"work-meeting"* are *"meeting"*, *"termin"*, *"n-abstracts"* and *"n-concepts"*. This organization makes it possible to formulate selectional restrictions within the integrator depending on more general types without being forced to specify either a complete list of all ground types or to introduce artificial features that describe certain common properties of ground types.

The source representation of a lexicon consists of a list of surface representations and the corresponding feature structures. The syntax of the feature structures is described in Table 3.2 in Section 3.3.2. Additionally, we implemented parame-

terized macros that allow for a flexible form of text replacement. Macros can be recursively embedded. This surface representation of a lexicon is compiled into an internal form that additionally allows a fast retrieval of lexical entries using an index realized using red-black trees (cf. Cormen, Leiserson and Rivest, 1990). One of the main properties of the knowledge sources used in MILC is the use of one single type lattice for all purposes. This renders a compatibility of the feature structures in the system, regardless of which component created it.

The direction of processing within the module for partial parsing is strictly from left to right. This entails that a cycle of agenda execution is carried out each time a new input fragment reaches the module that covers a vertex being farther right in time than the currently active vertex. This cycle first evaluates the scores of all tasks on the global agenda and carries out a beam search by selecting all tasks above the current threshold. Depending on the configuration of the parser the width of the beam can be regulated. Besides exactly specifying the width, two additional methods of selecting a width are provided: The user may choose to execute a certain percentage of the tasks in the agenda (e.g. 60% of the tasks), or he could specify a certain maximum number of tasks that should be carried out in each cycle (a typical value would be 200, allowing at most 200 tasks per chart vertex). After setting the threshold, the remaining tasks are considered one after the other.

Each task consists of the execution of one of the basic functions of chart analysis. The two important operations we use are the proposal of new categories using rules from the grammar, and the combination of an active and an inactive edge.

The rules of the grammar are formulated as feature structures of the type *rule*. The feature rule constitutes a list of embedded feature structures, the first of which is the left hand side of the rule. The remaining feature structures in that list form the right-hand side of the rule. Thus, the analysis uses a schema oriented at phrase structure rules, in contrast to methods that opt for a principle-based account of syntax (cf. e.g. Konieczny, 1996). In fact, the grammars within MILC loosely resemble those of PATR II (Shieber, 1984) by mostly providing a context-free skeleton (given by the types of the feature structures of a rule) and annotations to each element in the rule. Figure 4.13 depicts what a syntactic rule for partial parsing in MILC looks like.

This grammar rule licenses nominal phrases that are built from a determiner (a feature structure with the type *det*) and further nominal parts (type *n2*, in the simplest case a noun). The syntactic and semantic structure is percolated up from the noun to the NP, which corresponds to the *Head Feature Principle* of the HPSG. Inheritance is specified by coreferences between the subordinated elements on the right-hand side of a rule and the feature structure representing the left hand side. The determiner in front of the NP decides about the definiteness of the NP. Additionally, agreement with other parts of the NP is demanded.

The grammar is always consulted when an inactive edge is inserted into the chart. It is examined if the feature structure annotated at the inactive edge is unifiable with the first substructure on the right-hand side of any rule. For each such rule, a new edge is created that takes the result of the unification as a label. The

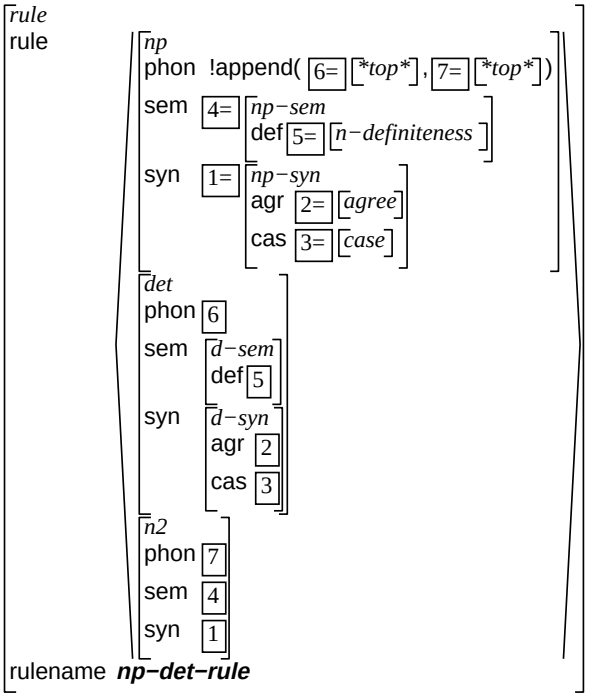


Figure 4.13. A grammar rule for noun phrases with a determiner

new edge is active if further constituents are required on the right-hand side of the rule. If, however, the rule has been consumed completely (under the circumstances described here this would have been a chaining rule with only one element on the right-hand side), an inactive edge is created. A new task to insert the edge into the chart is generated and put on the global agenda.

The mode in which the grammar is used in the proposal of new edges shows that MILC uses a bottom-up strategy for partial parsing, or more exactly, a strategy described by Kilbury (1985) (bottom-up left-corner parsing) that avoids the insertion of empty active edges, thereby ensuring that the chart does not contain trivial cycles. Common processing strategies can be divided into top-down and bottom-up strategies. An overview of several different approaches and their properties is e.g. given by Wirén (1988), who points to relations between parsing strategies and reason maintenance systems in the framework of incremental processing of written language in his dissertation (Wirén, 1992).

The second main operation we want to discuss here is the combination of two edges of the chart. The combination is carried out if an active and an inactive edge have a common vertex. In the framework of parsing this means that the intersection of the end vertices of the active edge and the start vertices of the inactive edge is not empty. If this is the case, then the system tries to unify the label of the inactive edge with the substructure of the active edge that specifies the next symbol within the active grammar rule. If this succeeds, a new edge can be created. Depending on the position of the active substructures within the rule, the newly created edge is either active or inactive. If it is inactive, the left-hand side of the rule has to be extracted to function as label of the new edge. Consequently, the grammar is searched for matching rules which could be used in the bottom-up step described above.

As previously mentioned, the partial parsing stage is strictly incremental and works from left to right. Thus, the condition stated here is the only one which may lead to the combination of edges.

The score for the new edge is computed from the scores of the originating edges. The acoustic score is calculated as length-normalized optimal mean of the acoustic scores of the combined edges (see Section 2.6.3). Additionally, we calculate a modifying score based on a stochastic language model. This model predicts the probability of a sequence of words with regard to their adjacency. The source we use for this computation is the model most often used in Verbmobil, capable of calculating bigram probabilities. The language model score is normalized according to the number of words in a sequence.

The output of the partial parsing component finally consists mainly of inactive edges which have goal status according to the type hierarchy. The user defines one type of the lattice to be the goal type; all edges annotated with a feature structure having a type that is subsumed by that goal type are delivered to the integrator and the transfer modules. Figure 4.14 shows an example for such a result. It demonstrates the feature structure transmitted as the partial parsing result of the noun phrase “das nächste Arbeitstreffen” (“the next work meeting”). In ad-

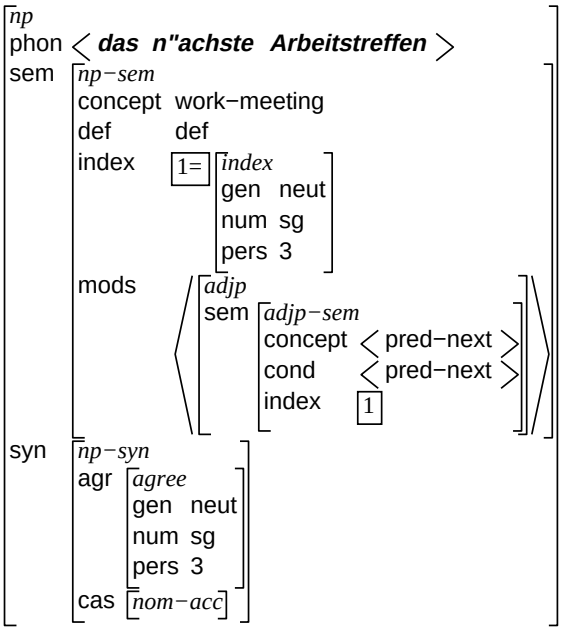


Figure 4.14. A noun phrase from the partial parser

dition, inheritance information with regard to basic word hyperedges and already recognized idioms are sent out, together with inhibitions that belong to idioms.

4.7 Utterance Integration

The task of the utterance integration component of MILC is to create, from the partial analyses that were given to it by preceding modules, semantic descriptions that are preferably complete and span larger subparts of an input. This includes in particular the construction of verbal phrases, but also the attachment of adjuncts to noun phrases, e.g. of prepositional phrases. Thus, the syntactic-semantic analysis, which has been started by the partial parser, is completed by the utterance integration.

The input data that have to be handled by the integrator include:

- **Inactive hyperedges from the parser** that have been assigned goal status. Noun phrases, prepositional phrases, adverbial phrases, etc. have goal status. Additionally, this includes verbs for which only a lexical access has been carried out and which are handled solely by the integrator. These edges are inserted into the chart and will be combined using the algorithm described below.
- **Modifications of hyperedges** that are treated analogously as explained for the partial parser. Modifications may occur in the form of additions to vertex sets, the augmentation of acoustic scores or the inhibition of word edges due to incorporation into an idiom. All three types of modifications are inherited within the component of the integrator. Moreover, if needed, they are passed on to the transfer component.

The main knowledge source used by the integrator is its grammar, which tries to combine partial analyses delivered by the parser to bigger, longer descriptions. In contrast to the parser, however, the integrator is capable of carrying out its task using island analysis (cf. Steel and Roeck, 1987; Stock, Falcone and Insinnamo, 1988). This algorithm does not assume that rules of a grammar have to be worked on strictly from left to right. This means that during an island analysis, the first element on the right-hand side of a rule is not always consumed first. The mechanism to control the application of grammar rules is done by the definition of islands within rules. In each rule one symbol (in our case always a feature structure) is defined to be the *island* of that rule. This element is always treated first. After finding a derivation for the island symbol other symbols to the left or right of the island may be analyzed. For instance, the rule in Figure 4.15[26] describes a nominal phrase, consisting of a determiner, an adjective and a noun. The main restriction that is formulated by this rule for analysis purposes is that the adjective is consumed first. After that, the order is not specified, depending on the processing strategy the determiner or the noun may be the next elements to be integrated into an analysis.

[26] This rule is for illustration purposes only. Within MILC, adjectival complexes are not constructed by the integrator; instead, they are handled during partial parsing.

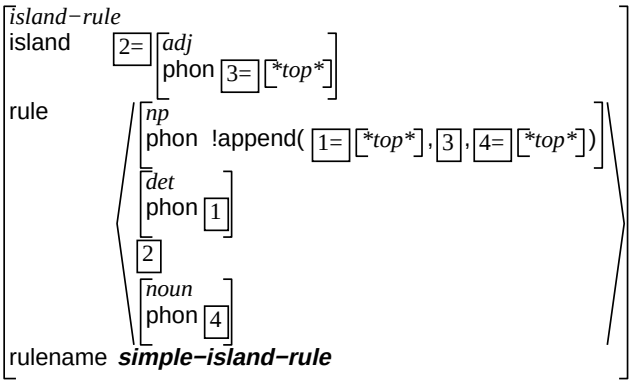


Figure 4.15. Island analysis: An example rule for nominal phrases

The motivation for such an approach to the structural analysis of utterance is at least twofold:

- Parsing of input data should start using partial descriptions that render the most extensive restrictions possible concerning the combination of elements of the input. This helps reduce the analysis effort and thereby increases efficiency. This motive dominates the approaches for bidirectional chart analysis of written input, cf., e.g., Satta and Stock (1989) or Satta and Stock (1994). In particular, the introduction of a head-driven analysis strategy is feasible in an elegant way (Nederhof and Satta, 1994). However, using an island parser will still deliver the same results as a conventional modeling. Only the search for results will be done in a different, better way (Stock, 1989).
- Parsing of input should start using partial descriptions that are the most promising to deliver results. This argument has been applied mainly for the analysis of spoken language. Word hypotheses are promising if they carry a good acoustic score, i.e. those that have been recognized with a relatively high degree of reliability. The primary idea is to pick out these "secure islands" and, starting from them, to extend to the left and right, picking words that are compatible with the already analyzed words and to integrate them into a common context (cf. Brietzmann, 1992). As the parsing of spoken language usually tries to gain efficiency by cutting off parts of the search space, the use of an island analysis schema may result in pruning different parts of the search space rather than in the conventional analysis. Therefore, the results delivered by an island analysis may not be equivalent to the results of a conventional analysis.

At first glance, it seems to be counter-productive to use a bidirectional parsing schema given the framework in which the parsing will happen, namely the incremental (time-synchronous) analysis of spoken language. However, to use such an argument in a dogmatic fashion would neglect some important aspects. The main motivation for introducing an island approach for utterance integration has already been stated: To prevent the construction of unnecessary complement complexes. Moreover, it does not seem reasonable to hypothesize the occurrence of optional elements of constituents (be they optional complements or adjuncts) regardless of their factual appearance. In this respect, island operations help by doing not more than is actually needed.

All these measures target an augmentation of the efficiency of the parsing process and thus ultimately an augmentation of the accuracy, as fewer unnecessary tasks will be created within the current search beam. Acoustically worse, but structurally better, tasks can be regarded during the processing of an agenda. In principle, however, the question would be how compatible the island analysis is with an incremental paradigm, or how much of the incrementality is lost using it. What needs to be considered here is the delay in time that results from using an island parser in contrast to using a strict left-to-right approach. The delay in question is mainly due to the fact that the module waits until a verb is present before it begins to construct complement complexes. This delay is surprisingly small, only a few tasks. These tasks have to be carried out after inserting the verb into the chart and consist of the saturation of the subcategorization list of the verbs using complements that appear to the left of the verb. In the case of German matrix sentences, there is only the sentence subject, which makes a combination of edges necessary. The extreme case is given with German subordinate sentences with a final verb position. In this case the attachment of all complements of the verb has to be carried out, which results in one additional task per complement. The rule describing this phase of the analysis is shown in Figure 4.16. Additionally, a chaining rule has to be used to insert the completed verb phrase into the chart. The additional effort for using an island approach thus consists of one to approximately five tasks.

Using a bidirectional approach renders one more consequence: The agenda is no longer bound to a specific point in time. In the case of partial parsing a processing cycle of the agenda is initiated for each step in time. Tasks that work on earlier points in time only surface if the acoustic score of an edge is modified and there is a prior task that reaches above the beam threshold after this modification. In the integrator, the association of a task with some point in time is practically meaningless. Even here, the agenda is evaluated and tasks are carried out whenever a new vertex for the analysis is created. However, the tasks themselves might relate to arbitrarily different points in time.

In order to make the proceeds of the analysis more clear, we will now give a short description of the analysis of “lassen Sie uns den nächsten Termin ausmachen” (“let us schedule the next appointment”). The more interesting messages that reach the integrator, coming from the partial parser, are shown in Table 4.3 on page 125.

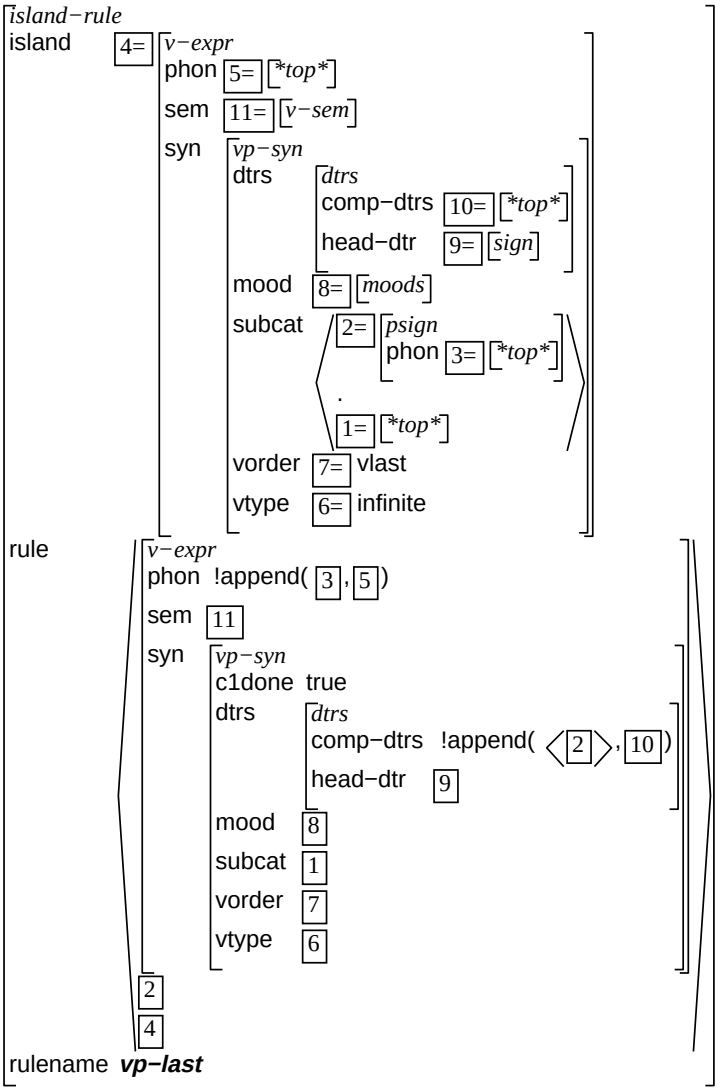


Figure 4.16. A rule for complements to the left of the verb

Table 4.3. Sketch of the processing of "lassen Sie uns den nächsten Termin ausmachen" ("let us schedule the next appointment")

Nr.	Content	Translation	Comment
1	lassen	let	Verbs are passed on to the integrator from the parser
2	Sie	you	NP
3	uns	us	NP
4	Termin	appointment	NP
5	nächsten Termin	next appointment	NP
6	den nächsten Termin	the next appointment	NP
7	ausmachen	schedule	Verb

First, the imperative reading of "lassen" is inserted into the chart (we neglect the other possible readings). It subcategorizes for three complements, namely the subject, a direct object and a subordinate verbal phrase using an infinitive in final position. After inserting the imperative, a grammar rule demanding the initial position for those can be applied. The result is an active edge with the type *v-expr* that spans the verb and has a second element on the right-hand side of the rule. This stems from the fact that we saturate the complements in a subcategorization list one after the other. The two following edges arriving at the integrator ("Sie" and "uns") are inserted into the chart and bind the first two complements. The longest active edge is now a description of "lassen Sie uns" which waits for a verbal phrase.

The next three edges "Termin", "nächsten Termin" and "den nächsten Termin" are of no immediate importance for the outstanding analysis, they are only introduced into the chart. The last message arriving at the integrator is the description of "ausmachen". The corresponding feature structure is shown in Figure 4.17.

The grammar rule shown in Figure 4.16 on page 124 takes the verb and produces a verbal expression that only contains the object in its subcategorization list. The insertion into the chart thus results in an active edge. The annotated feature structure has the type *rule*, because the corresponding grammar rule has not been completed yet.[27] The right side of the rule contains structures of the types *np* and *v-island* (*v-island* is a subtype of *v-expr*). The edge carries a notice that the first element has not yet been consumed.

This schema results in a situation in which the system has to look for matching inactive edges to the *left* of an active edge just inserted. In this case, the module finds three possible candidates: The nominal phrases just mentioned. Consequently, three inactive edges can be created. However, only one of them, the longest one, is

[27] The annotations of active edges are subsumed by the rule that has been applied. Inactive edges only receive the left side of a rule as annotations.

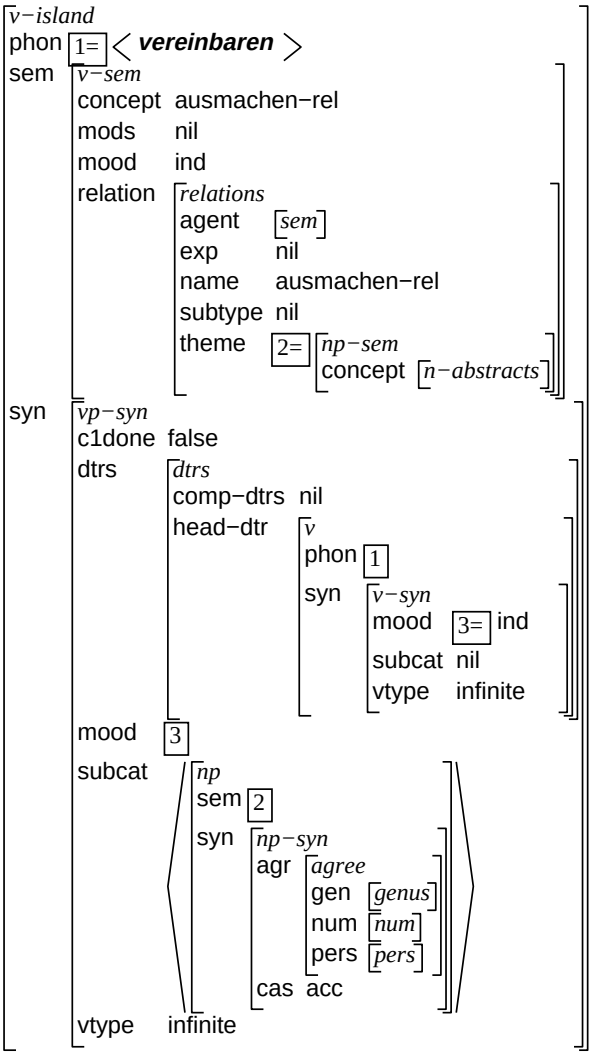


Figure 4.17. Lexicon entry for “ausmachen” (“schedule”)

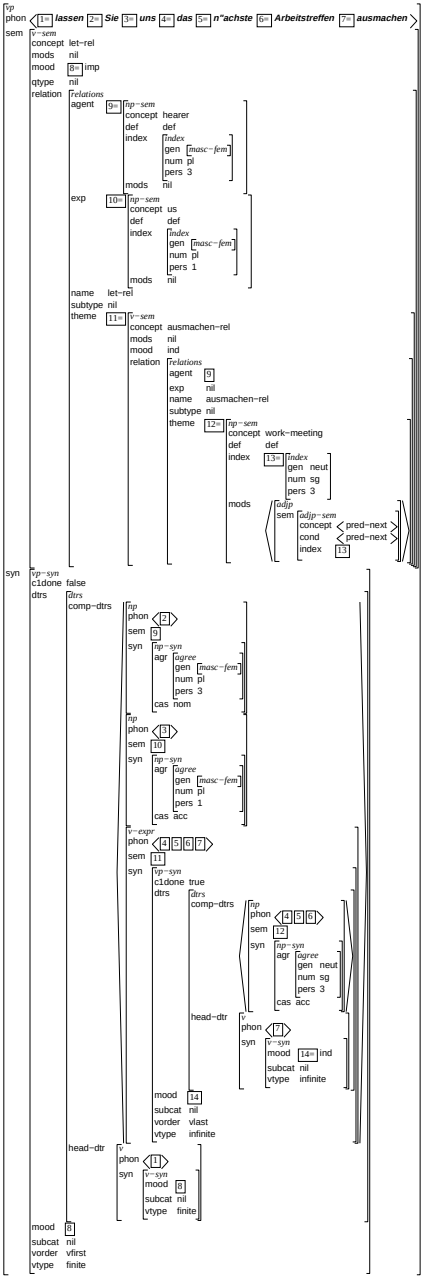


Figure 4.18. A verbal phrase from the integrator

adjacent to an active edge with which it could be combined. After this is done, the inactive edge depicted in Figure 4.18 on page 127 can finally be created, spanning the whole relevant input.

If this edge has goal status, which again is examined using subsumption with a predefined type, it is sent to transfer. Besides the delivery of these completed analyses, the utterance integration also passes along inheritance information.

4.8 Transfer

In principle, the task taken over by transfer in a machine translation system is to link linguistic descriptions for different languages. As already mentioned in the introduction, to use transfer as opposed to interlingua-based approaches has become more and more popular. This is particularly true if a system is not going to be used for utterances in small, technical domains, but if it has to process common speech, which is much harder to grasp for interlingual applications (Hutchins, 1994). Moreover, a transfer-based translation method must be preferred if spoken language has to be processed. Additionally, transfer may be characterized as a process model, as opposed to the static description of linguistic knowledge using an interlingua. However, we have to note that in most systems the transfer stage is still mostly a recipient of information without having major influence on the production of that information by the preceding components in a system. In principle, a transfer model is imaginable that represents the central component of a MT system. Beginning on a high abstract level, that module could issue requests to other knowledge sources. These requests are iteratively decomposed into more simple requests that finally lead to expectations that are evaluated e.g. in the speech recognition module.[28]

This seems not to be plausible from a psycholinguistic viewpoint and can probably not be implemented successfully. In any case, transfer may add to the disambiguation of the input. It should be possible to prevent further processing on hypotheses that have been classified as not transferable, and it should even be possible to incorporate predictions that have been generated by a transfer component. In addition to this, a transfer component could initiate further steps of evaluation by other components if this seems necessary because of lack of sufficient information or a translation quality that is too low. For instance, such an approach is promising if the time systems in two languages diverge (Eberle, Kasper and Rohrer, 1992), or it could be used for ambiguous deictic references: "Geht es da bei Ihnen" may be interpreted as a temporal reference ("Is that possible for you") or as a local reference ("Would it be okay at your place") (Alexandersson, Reithinger and Maier, 1997).

[28] A system emphasizing the role of transfer is TDMT (Sobashima *et al.*, 1994). TDMT produces translations on the basis of previously seen examples. The most important knowledge source is a table of transfer entries that are compared to the surface representation of the actual input. If necessary, further evaluation processes, lexical or syntactic, are initiated. Moreover, cf. Kay, Gawron and Norvig (1991, S. 89ff).

Such additional evaluations are already put to work in today's machine translation systems, e.g. to treat lexical mismatches (Dorr, 1993) in the case of overgeneralizing word correspondences or to guarantee a correct translation of tense and aspect in certain cases (Buschbeck-Wolf, 1995). The organization of requests for additional information and the handling of the replies to such requests can easily be achieved using a layered chart, because it offers the exchange of consistent information using edge references and thus does not need a complex question-answer protocol with advanced delay mechanisms.

The situation in which transfer is applied in common translation systems today is the following: A source language input utterance is present in a completely analyzed form. The result of the analysis is a structure containing a description of the input on a syntactic or semantic level. Usually the systems request the description to be unique; multiple analyses that cover different structural or content structures are treated by applying the transfer apparatus iteratively. The input structure is traversed top-down, a target language specification is constructed beginning with the top node. Within the search space of transfer, a breadth-first search is carried out; the next level of the input representation is only used after all higher levels have been completely analyzed. In order to analyze written input, such a conservative mechanism might be enough to succeed; however, the incremental processing of spoken language demands a more elaborate procedure. This is already clear from the fact that a complete view on the input is only possible relatively late during the processing. Each component has to begin before that point in order to produce partial results.

Regarding the architecture of transfer, the first thing we have to note is that the introduction of a separate component seems to be adequate. Even if the principle argument stemming from the weak modularity hypothesis and global insights from software management in Chapter 1 is not taken to be reason enough, there are still linguistic motives for such a modularization:

- Natural language processing systems are in many cases oriented towards the classical linguistic levels of description, morphology, syntax, semantics and pragmatics. Thus, for each level there is one module in a system. The annotation of rules of one of these components with information about how to carry out transfer is possible. But to tie transfer knowledge to any of the other levels contradicts the intuition of different processing steps that seem at least to be partially evident, cf. Chapter 1. Moreover, language understanding and translation are separated within human information processing, although both processes may overlap. Also, the concept of a variable depth of analysis suggests not to bind transfer rules to rules of any other level: For instance, the annotation of transfer knowledge to syntactic rules can neither cope with lexicalized translations (idioms) nor with the semantic problems requiring a deep analysis (Prahl *et al.*, 1995).
- The correspondence between transfer rules on one hand and rules of any other module on the other hand is not symmetric. Let us again take syntactic analysis as an example. It seems to be clear that not every structure-building rule in a syntactic grammar corresponds to a rule that would be able to translate the struc-

ture thus created.[29] For instance, an English prepositional phrase like the one embedded in "Mounting of the drill" might be translated into German again as a PP "Montage von dem Bohrer" or, stylistically better, as a genitive attribute "Montage des Bohrers".[30] There is no reason to already include both transfer possibilities at the time of constructing the prepositional phrase within a parser, as the genitival translation is a special case. Additionally, the creation of syntactic structure and transfer do not work synchronously. During the processing of "the big dog", the syntactic structure can be built gradually, going from left to right. The translation, however, must be delayed until the head of the phrase is known to the system; for one in order to get the correct gender of the noun phrase and to produce an adequate determiner, but also in order to translate the adjective correctly ("big" having the meaning tall in stature vs. being important).

4.8.1 Chart-Based Transfer

Until now, charts have been used for analysis and generation for the most part. It turns out, however, that this kind of data structure — and, thus, simultaneously the advantageous division between algorithm schema and search strategy — can also be used for transfer. There are many analogies between the transfer in a mainly symbolic approach to translation based on feature structures and a syntactic structural analysis. Partial translations of a structural description of some input can be potentially useful for several of the steps during transfer. Therefore, it is reasonable to store these in some kind of well-formed substring table. The extension of this concept in the direction of a chart is possible without any problems: Inactive transfer edges describe complete translations of input constituents, while active transfer edges belong to partial translations.

Of course, there are other approaches to the translation of natural languages, besides methods oriented at such a strong symbolic paradigm. The most prominent member of these is the statistical translation (Brown *et al.*, 1990). The approach is to try to derive probabilities of correspondences between sections of texts in two languages by comparing several examples of bilingual texts. The most probable correspondence of a source language text, given a model for mapping from one language to another, is presented as its translation. The biggest problem in this domain is to map words and phrases correctly to each other during the training phase (cf. e.g. Gale and Church, 1991; Fung and Church, 1994). A high degree of complexity is given by lexemes consisting of multiple words that may be represented discontinually in some target language. Moreover, the mapping has to account for differences in phrase length in different languages (Wu, 1995a). Melamed (1998) thus completely neglects any supra-lexeme correspondences and assumes that usually one word in one language is translated as exactly one word in another language.

[29] Despite these observations, there are approaches that annotate syntactic rules with translations (Katoh and Aizawa, 1994). However, the grammar is partitioned in order to equip, for example, fixed expressions directly with a translation.

[30] This example was adapted from Beskow (1993).

Errors, which occur inevitably, are smoothed using a noise model. Moreover, he uses knowledge of word classes (open class vs. closed class) in order to augment the precision of translations. He reaches success rates of more than 90% for translations of extracts from online Bible texts.

The work of Melamed (1998) exemplifies the trend of not completely relying on a statistical modeling of text equivalence on the basis of surface representations. This trend is also explored in other projects. The degree to which conventional language knowledge is employed in a system ranges from using a grammar to augment the statistical component (Wu, 1995b) over the use of knowledge from both domains to similar degrees (Knight *et al.*, 1994; Rayner and Carter, 1997), to the use of a statistical language model to augment the output quality of a symbolic machine translation system (Brown and Frederking, 1995). For some time now, there have also been attempts to use statistical approaches for the translation of spoken language (e.g. Vogel, Hahn and Branigan, 1996; Niessen *et al.*, 1998), or to experiment with connectionist methods (e.g. Wang and Waibel, 1995).

A different corpus-based approach uses knowledge about translations already carried out to simplify future processing (so called *memory-* or *example-based* translations) (Sato and Nagao, 1990). The main knowledge source for such systems is a large database of translations that can be acquired beforehand from bilingual corpora. In contrast to purely statistical models, example-driven models seem much more flexible. For one, new translation pairs can be easily added without having to reestimate transfer probabilities. Second, memory-based models can be extended in a more abstract way by not only having pairs of surface representations as a basis for translation, but also for instance allowing patterns with variables that would represent trivial, non-recursive transfer rules (Juola, 1994). Moreover, an integration with other levels of linguistic descriptions is possible this way (Furuse and Iida, 1992).

In our view, hybrid paradigms for translation of spoken language are the most promising. Having a symbolic component, they guarantee that previously unseen input can be processed. At the same time, they treat frequent contrastive collocations in an efficient way. However, in the framework presented here, we will restrict ourselves to a purely symbolic approach.

A preliminary investigation (Amtrup, 1995a) models the structural transfer using syntactic descriptions. The transfer chart used there is an extension of the chart used for parsing (see Figure 4.19). Each inactive analysis edge that contains a complete syntactic description of an input constituent is taken to be the root of a tree of transfer edges. The first level of nodes in such a tree is solely created by applying transfer rules representing contrastive grammatical knowledge. Further levels are constructed by incorporating already existing partial translations. This process is initiated by the existence of subordinate transfer equations. This incorporation constitutes the fundamental rule of transfer, i.e. the creation of a new edge from an active edge (which still contains unsolved requests for recursive transfer) and an inactive edge (which represents the complete translation of a constituent).

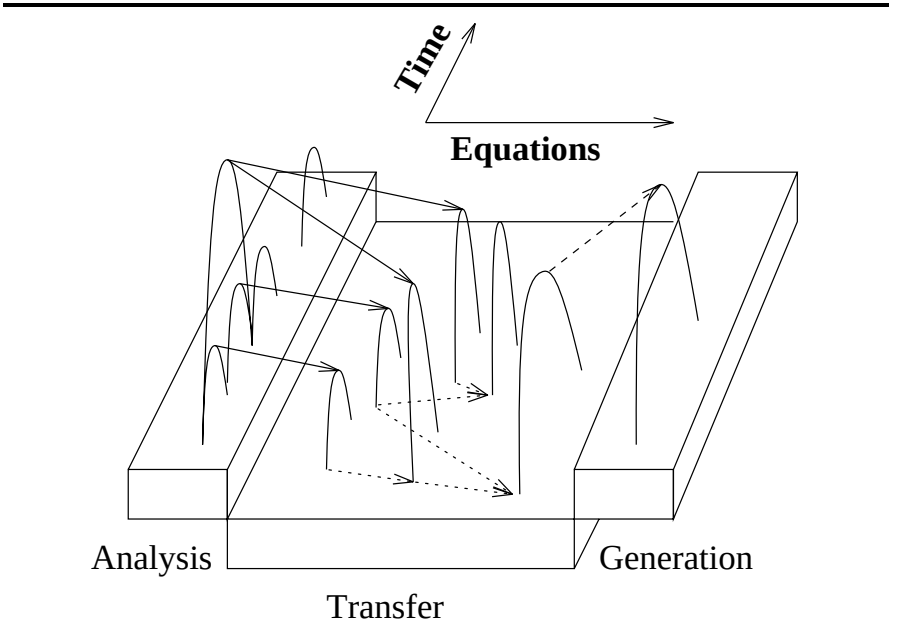


Figure 4.19. A transfer chart for structural translations

Thus, such a transfer chart realizes a two-dimensional structure. One dimension describes the extension of the underlying input in time, the other dimension represents the number of unsolved transfer equations.

4.8.2 The Implementation of Transfer for MILC

The approach for chart-based transfer chosen for the work presented here is in two respects a generalization of the method just described. First, the description of translation knowledge is not only based on syntactic criteria but also incorporates semantic knowledge to a great degree. Secondly, we abandon the rigid adjacency requirement, which is adequate for a transfer component based on syntactic structural rules, but that is not sufficient for the syntactic-semantic descriptions that show up in MILC.

The input for transfer consists of inactive analysis edges from the integrator, together with inheritance and inhibition information. The edges contain feature structures that describe the syntactic-semantic content of an input constituent. The principle goal is to find a corresponding English description for that input.

The first step is to find applicable transfer rules for the given input structure. An example of such a rule is shown in Figure 4.20. In analogy to some of the meth-

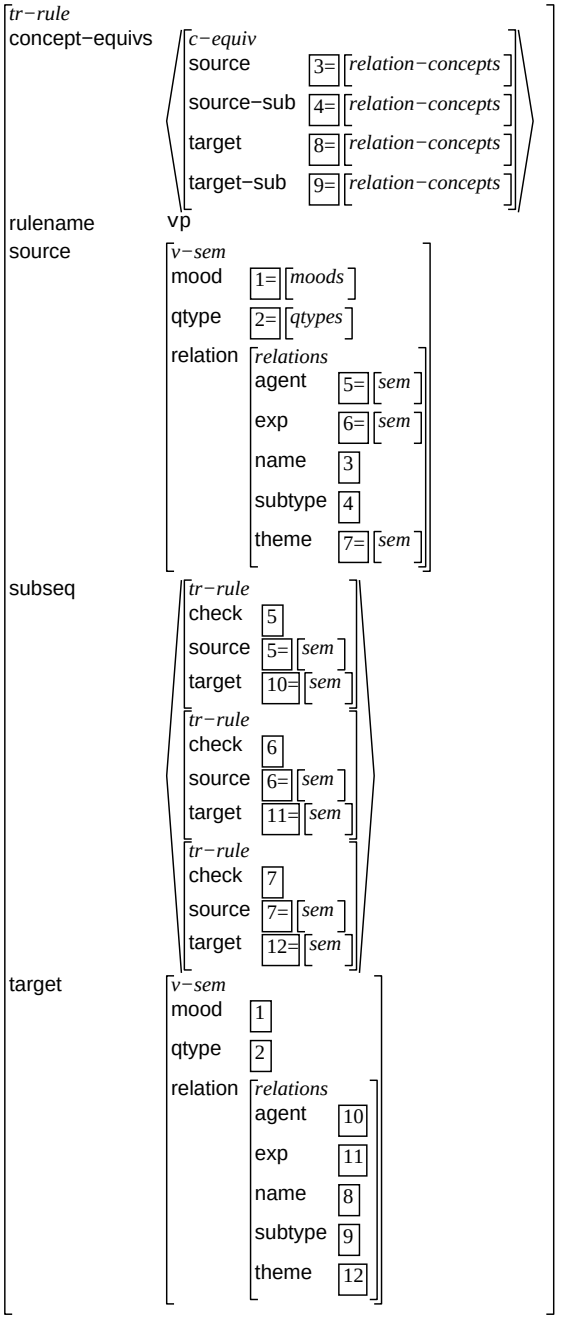


Figure 4.20. A transfer rule for verbal phrases

c−equiv	
source	sein−rel
source−sub	pred−recht
target	suit−rel
target−sub	nil

Figure 4.21. The transfer lexicon entry for "recht sein" ("suit")

ods explained in Section 3.2, a rule consists of two central features, one of which (source) contains the source language semantic description, and the other (target) the corresponding description in terms of the target language semantics. Contrastive equivalences between these two are either expressed using direct coreference (feature qtype in Figure 4.20) or indirectly (features subseq and concept-equivs). Here, the subseq feature defines the necessary partial translations that have to be carried out recursively. Their use is analogous to the τ-relations of Kaplan *et al.* (1989) in that a correspondence is demanded on different sides of the translation. The feature concept-equivs mainly drives the transfer of lexical concepts. From the example it should be obvious that the filler of case roles (agent, experiencer, theme) are treated using recursive transfer, while the main concept of the relation is controlled using a regress to a transfer lexicon. The elements of that lexicon map German semantic concepts to their English counterparts, as shown by Figure 4.21.[31]

The first part of the application of a transfer rule consists of the unification of the source feature of the rule and the semantic aspect of the incoming analysis feature structure. Hence, for instance the main distinctions between verbal and nominal categories are drawn. Moreover, the coreferent contents in the target language description as well as the source language parts of conceptual equivalences and subordinate transfer equations are set. In case of a successful unification, the result is a transfer edge, the status of which (active or inactive) has yet to be determined. A transfer edge is said to be active if it contains one or more subordinate transfer equations that are open, i.e. still unsolved. This classification corresponds to the use of the notion of activeness in chart parsing approaches and in the preceding syntactic-driven research. In order to keep the number of transfer rules small and to be able to formulate general rules, we do not require that every equation of the subseq-feature in a rule is accounted for. In contrast, after the unification has happened, all elements of that list are examined in order to find out if they are relevant in the present situation or not. If the source structure of an element of the subseq list is not used, it is

[31] To be more precise, the entries map syntactic representations of a semantic concept that are formulated close to the German language in a form that is more suitable for the English generation.

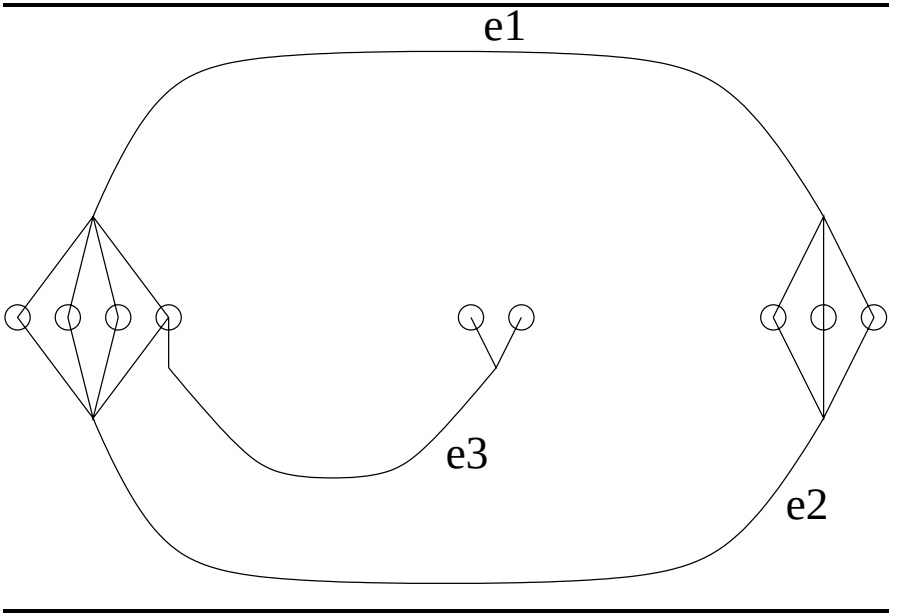


Figure 4.22. Topology of the fundamental rule for transfer

deleted from the list. The categorization of whether an edge is active or inactive, is only carried out after this step.

Now, the fundamental rule of chart transfer is to combine an active and an inactive transfer edge. A pair of edges basically has to fulfill two conditions in order to be processed.

They have to be compatible with respect to the topology of the graph. In the case of the syntactic transfer chart this meant that the active and inactive edges had to be adjacent. This condition could always be met because the complete partial graph that was built from analysis edges was present in transfer. The situation is different, however, for the implementation described here. Only edges that had goal status according to the integrator are delivered to the transfer component. This entails that for several inactive edges no sub-paths are present that could be used for a simple compositional transfer using a bottom-up strategy. Moreover, some modal particles are treated as artifacts of spontaneous speech. They are integrated into the analysis in order to be able to create structures spanning them, but they do not contribute to the semantic content of an analysis.

One obvious solution would be to move away from a chart-based processing mechanism in favor of a conventional transfer approach. That decision, however, would neglect the fact that even under the circumstances just described, a desirable reuse of already constructed partial translations can be carried out. Additionally, one

would lose the advantageous control mechanism provided by the use of an agenda in a chart-based paradigm.

Thus, we preferred to use an approach keeping the advantageous chart properties. We do not require that one of the end vertices of the active edge be identical to one of the start vertices of the inactive edge, thereby extending active edges by consuming inactive edges adjacent to them. Instead, we leave the sets of start and end vertices as they were given by the inactive edge triggering the creation of the active edge. The relation between two edges in transfer has to be the following: The active edge spans the inactive edge with which it is to be combined. Since transfer edges are hyperedges as is the case in the rest of the system, we use the earliest start vertex ($\alpha_<(e)$) and the latest end vertex ($\beta_>(e)$) of both edges to check this. Figure 4.22 shows a possible arrangement of edges. **e1** depicts an inactive analysis edge that has been delivered to transfer from the integrator. The edge **e2** was created due to the application of a transfer rule and thus has identical start and end vertices. **e3** denotes an inactive transfer edge which has been created beforehand. It is completely covered by **e2** and therefore can be combined with that edge.

This type of evaluation does not take into account the fine-grained topology of the graph. Thus, we cannot guarantee that an inactive edge that is incorporated by an active edge has been, in fact, a member of the sub-path described by the active edge. However, this is not critical, since almost all invalid cases are excluded by the following unification, which is the second condition for the combination of two edges. The feature structure describing the inactive edge must fulfill one of the open transfer equations of the active edge, i.e. the unification of these must succeed. Thus, the compatibility of the semantic descriptions is ensured.

If no matching inactive edge can be found using this procedure, we have to rely on conventional recursive transfer. In order to do this, the open transfer equations that could not be solved are taken to be initial edges, which are handed to the grammar for a search of matching rules. Once the edges created by grammar rule applications are inserted into the chart, the normal operation cycle resumes.

Finally, for an inactive edge to be added to the chart, all concept equivalences have to be solved. Here, all combinations of transfer lexicon entries are considered that fulfill the elements of the concept-equivs feature. For each such combination of lexicon entries, the matching pairs are unified with the elements of the concept equivalences.

Edges that have been created that way are potential result candidates that might have to be delivered to the English surface generation. However, this only happens if the root of an inactive edge has been within the input from the integrator. The intention for this additional examination is to prevent the delivery of edges that have been proposed in the transfer component through recursive transfer. Those have not been marked as goal edges by the integrator and thus should not be given to the generation module. In total, the transfer module transmits only complete translations of constituents that have been relevant for the linguistic analysis.

4.9 Generation

Within the framework of MILC, we do not carry out a complete generation, but rather a surface realization if we argue in accordance with the classic division of the creation of a language form from a set of communicative goals (cf. Horacek, 1993). This segmentation consists of a strategic, content-generating text planning stage (the *"What To Say"*), a tactical text planning phase defining the form (the *"How To Say"*) and finally the surface generation proper. If we consider the input for the generator of MILC, this is reasonable, since they already are semantic descriptions of the content of single utterances that are quite near to the actual surface form. The planning of a lengthy discourse is thus not a topic for the research presented here. Even the part defining the surface form is quite lean in dimension; the input is underspecified concerning the lexemes that are generated, yet, we do not carry out a complete word choice. Instead, all words matching the input semantics are chosen.

After a generation system has accepted the semantic content to be verbalized, the order in which individual substructures are processed has to be stated. Moreover, a suitable representation for the partial results created during the generation has to be set up. In general, the realization of an algorithm schema (which defines the structure) and a strategy (which makes an algorithm from the schema) is necessary.

In the framework of generation, head-driven strategies are the ones used most often. It is quite natural to first select the lemma which dominates the content (*"semantic-head-driven"*, cf. Shieber *et al.*, 1989, Shieber *et al.*, 1990, Kikui, 1992) or the syntactic structure (*"syntactic-head-driven"*, cf. König, 1994) of an utterance. Starting out from that central information, additional language material can be placed into the correct position. This procedure terminates when all input elements have been considered (although not all input elements have to be realized overtly).

Concerning the embedding of a data structure for generation, it can be noted that chart-based approaches are often used to handle the operations and data elements needed. Chart-based approaches offer a non-redundant storage schema and the possibility of fine-grained control. Thus, not only is the repetitive generation of identical surface structures avoided, it is also possible to integrate different search strategies. For instance, the introduction of a bidirectional generation is possible, enabling the generator to construct islands within the surface realizations that are extended to the left and right (Haruno *et al.*, 1993). The only problematic issue is the fact that a direct mapping from chart vertices to points in time is not straightforward. The order in which different parts of a surface form are realized is created only during generation and thus cannot be used analogously to the analysis. For that reason, Kay (1996) uses sets of already processed semantic substructures as identification of edges.

A discrimination criterion for several different approaches to generation is also given by the kind in which the generator consumes its input. Most systems rely on the complete delivery of a semantic representation of the content before the generation takes place. This is in particular important for head-driven methods, since they have to know the head (be it semantic or syntactic) of an utterance in order to build

the structural frame of the surface form. Even chart-based systems that use flat semantic descriptions, like Kay (1996), are not specifically designed for incremental input. However, an incremental generation is reasonable and practical, as Reithinger (1992) demonstrates with the POPEL system (also cf. Finkler and Schauder, 1992, and Kilger, 1994).

One interesting variant of generation algorithms is the so-called *shake-and-bake*-generation (Beaven, 1992; Brew, 1992; Whitelock, 1992). Here, the input is considered to be a set of uncorrelated units. These units are combined to form bigger signs until finally all input units have been used and the sign thus formed conforms to the underlying grammar that licenses combinations. The main drawback of the early version of this system is its exponential complexity. Therefore, shake-and-bake generation has not been extensively used in applications. However, Poznański, Beaven and Whitelock (1995) publish a polynomial generation algorithm within this framework.

The generator for MILC combines some of the properties mentioned so far: It is incremental, chart-based and may operate on a head-driven basis. We will demonstrate the different aspects of processing using the example shown in Figure 4.23. It depicts a part of the input delivered from the transfer to the generator.

The feature structure in Figure 4.23 is only part of the input, as it only contains the semantic description covering the whole input. All analyses that represent a smaller part of the input are omitted here. The type of the feature structure (*v-sem*) instructs the generator to look for rules within its grammar that have the same type on the left-hand side.

The model underlying the derivation is again made up from phrase structure rules (see Figure 4.24). However, generation rules are treated differently from parsing rules. First and foremost, the direction of the application is opposite. In a structural analysis, the feature structures on the left-hand side of a rule are composed from subordinate structures on the right-hand side of the rule. The situation for generation is as follows: The semantic description for an element on the left-hand side of a rule is given as input to the system. The goal is to create the syntactic and phonological structure by iteratively processing the elements on the right-hand side. This enables a program to have a more restrictive method of selecting applicable rules, as it would have been possible simply by testing types of feature structures. The goal for having such a restrictive rule selection schema is to reduce the number of rule applications and thus the number of unifications. To accomplish this, the feature check shown in the rule of Figure 4.24 defines a path (or a list of paths) that must not carry empty values within the structure to be generated. For instance, this consideration is used to bar the generation of modifying phrases for a structure if the semantic input does not contain any modifying elements.

The author of a generation grammar has a second method of influencing the processing by modifying the order in which right-hand side elements in a rule occur. This order does not — as in analysis — specify the surface order of constituents. The surface order is defined by the phon feature on the left-hand side of a rule. This enables the author to arrange the elements on the right-hand side following

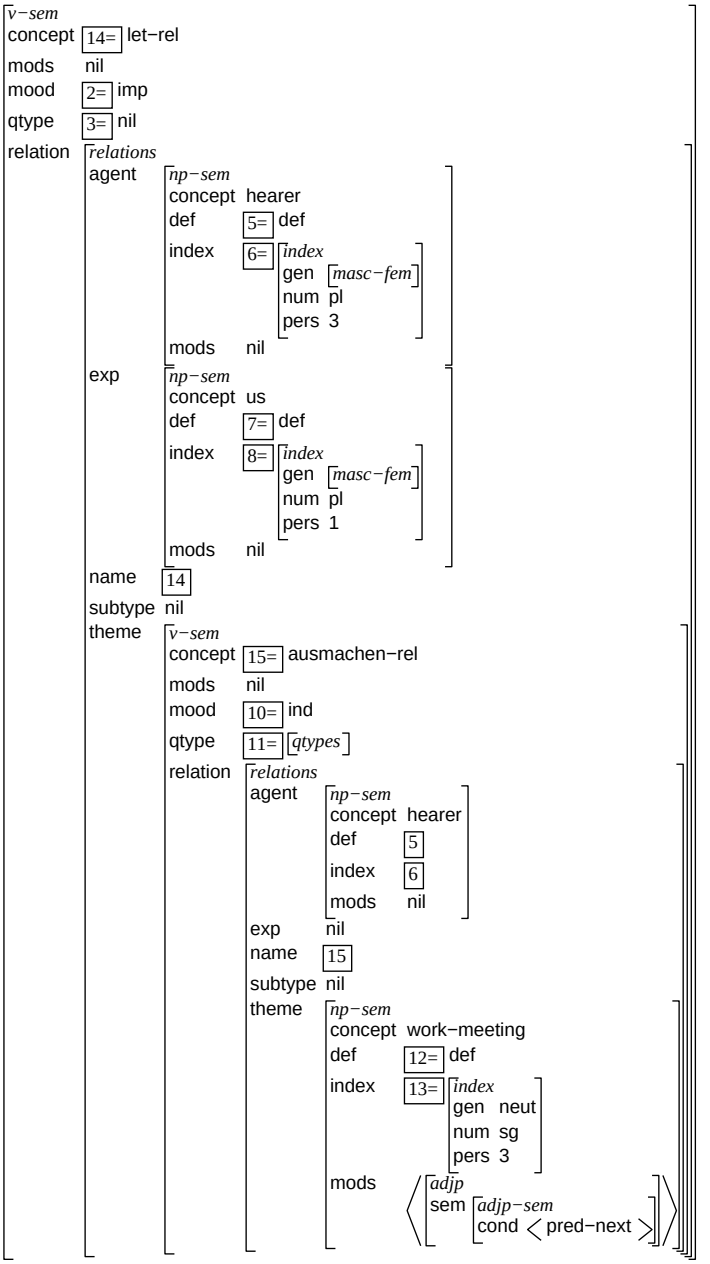


Figure 4.23. A subpart of the generation input for "lassen Sie uns das nächste Arbeitstreffen vereinbaren" ("let us schedule the next work meeting")

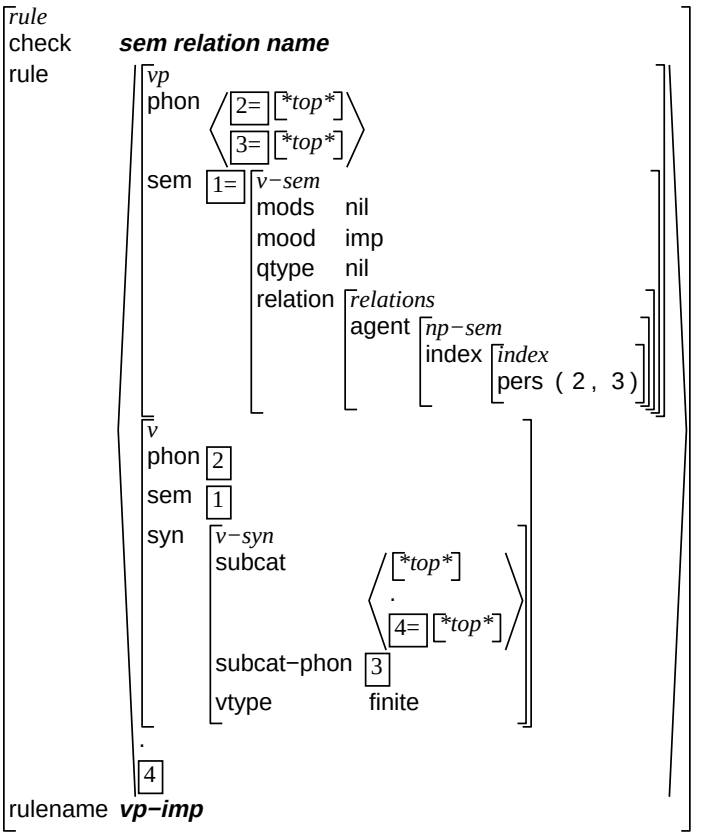


Figure 4.24. Generation rules for imperative verbs

economic principles. Usually, this means that the element that generates the most restrictions is generated first. For example, while generating a verb phrase, it is not reasonable to first generate the subject and then the finite verb form, since the verb may pose further agreement restrictions on the subject. Instead, the verb would be generated first, then the subject. Using this kind of economic ordering for the right-hand side of a rule, a head-driven approach can be easily integrated into the derivation.

A further advantage of the strategy presented here is the very general, elegant way in which rules for the subcategorization of verbs and other elements can be formulated. The formalism we use allows for the specification of a pair (which defines the head and the rest of a list, similar to the construction `[H|T]` in Prolog

or a Lisp cons-cell). Using this method, the subcategorization list of a verb can be used to dynamically create specific generation rules that contain exactly as many complements as the verb demands. This possibility is also used in the rule shown in Figure 4.24.

The generator that we present here works incrementally. However, the incrementality is not based on the steady flow of parts of a semantic descriptions into the generator, as it is done in the approach of Finkler (1996); instead, even in generation the basis of incrementality is the extension of input in time. The semantic descriptions usually arrive in the order of their endpoints. The mechanisms of chart-analysis are used to integrate already generated fragments into edges that cover bigger parts of the input utterance. Whenever an edge arrives in generation, coming from transfer, the generation grammar is consulted. It delivers rules that are potential candidates describing the surface realization of the given input. The search for rules is steered by the type of the fragment to be generated and by the features to be checked (stated by the check feature of a rule). If matching rules are found, their left-hand sides are unified with the input, respectively. The edges thus generated are inserted into the chart.

Depending on the number of elements that are still unaccounted for on the right-hand side of a generation rule, edges are classified as being active or inactive. The rules for active edges have not yet been completely satisfied. While adding such an edge to the chart, a search is carried out for inactive edges that are able to fill the rule element to be processed next. This is evaluated by unification. If successful, a new edge is created and undergoes the same cycle. Additionally, for a nonterminal on the right-hand side, the process of searching for applicable grammar rules is initiated again. This is due to the fact that there may be more derivations apart from the ones found until now. Using this mechanism, active and inactive edges are combined until finally one edge is found that completely realizes the semantic input description. In analogy, the generator tries to incorporate inactive edges within larger contexts. The implementation searches for active edges that span an inactive edge inserted into the chart and that are compatible with it. In this case, both may be combined.

The processing of terminal elements within grammar rules needs special consideration. A feature structure is called terminal if it is associated with a lexical type, like the substructure of type *v* in the rule shown in Figure 4.24. If such a rule element is to be generated, a lexical access is carried out on the basis of the semantic concept requested in the description. The generation lexicon is specified using full forms. We do not carry out morphological generation. For each matching word, a new edge is generated. Figure 4.25 shows a lexicon entry being relevant for the fragment used here as an example. The entry simultaneously shows the manner in which the order of the phonological values of subcategorized elements is computed. The list-valued feature subcat-phon describes the surface order of the utterance fragments belonging to the verb.

The special type *gener-cat* denotes edges that have to be realized by the generator. Those feature structures of edges created by the generator that are subsumed

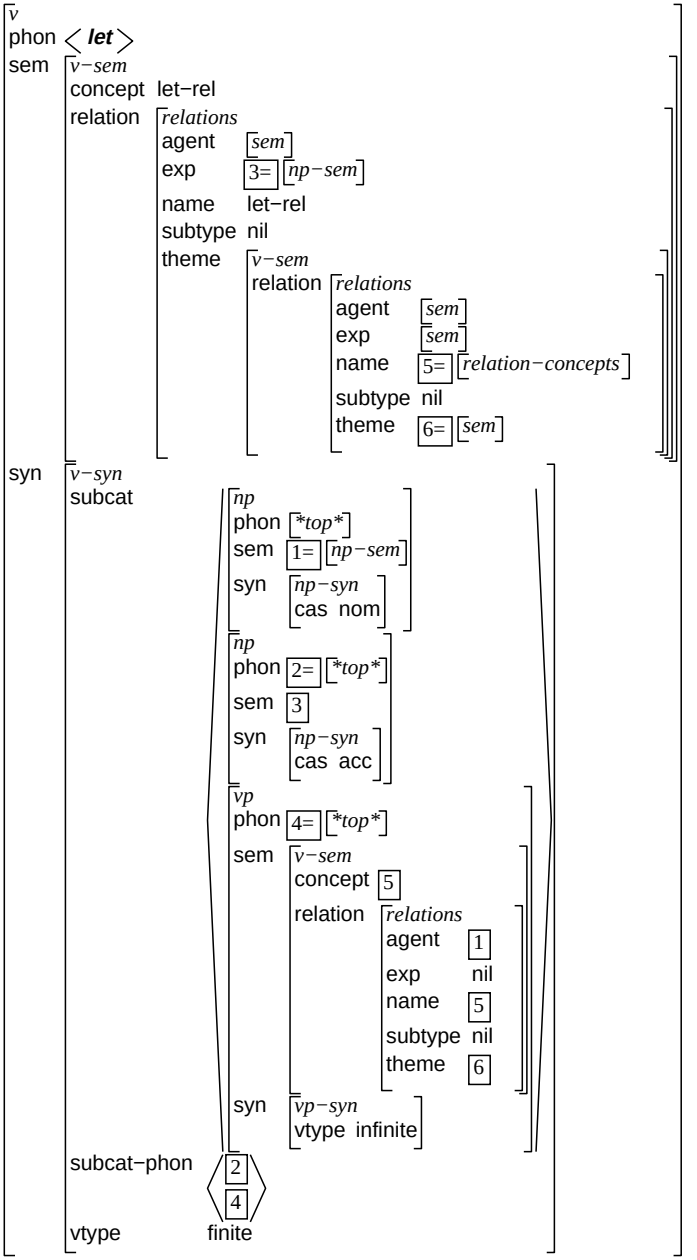


Figure 4.25. Generation lexicon entry for "let"

Table 4.4. An example for the output of the generator

you
you |we
you |we |it
you |we |it |work meeting
you |we |it |schedule work meeting
you |we |the next work meeting
you |we |schedule the next work meeting
let us schedule the next work meeting

by *gener-cat* are marked as final output of the system. The generation component maintains a hypergraph of these edges at all times. That graph represents fragments of surface realizations that could possibly be part of the translation of the input. Whenever a new edge is added to this graph, we use the incremental Algorithm 14 in Section 2.7 to determine the shortest path through the generation graph. If the path changes, the surface forms are printed as output. The weight of an edge is determined by the acoustic score of the original input words. Additionally, we impose a penalty for a transition between two vertices of the graph if the transition is not justified by an edge. Using this method, a path through the complete graph of surface edges is found. A further modification of the score depending on the length of the surface realization has the effect that semantic contributions are preferred to be realized overtly.

The method of keeping search information within the generator utilizes an incremental output of the target language utterance while it is being translated and generated. Table 4.4 presents the output of the complete MILC system for the German input "lassen Sie uns den nächsten Termin ausmachen", which was used for all examples in this chapter.

It is clear how evergrowing parts of the input utterance are translated into English. For instance, the imperative verb "lassen" ("let") is only generated after all subcategorized complements have been realized. Only using this schema the structural information of "den" being a determiner and not a demonstrative is incorporated.

4.10 Visualization

The visualization of information being created during the course of processing is essential for all large systems, be it for reasons of debugging, or to present results in a more instructing way than is possible with pages of textual representations. Several reasons for the importance of a visualization for MILC follow:

- MILC is a distributed system, currently consisting of six components for the analysis of an incoming speech signal. The search for programming errors in such distributed systems is notoriously complex; the uncertainty about the exact order in which some actions happen provides a whole array of error sources.
- MILC processes spoken language. This results in a very high number of initial elementary input elements (word hypotheses). Their number is reduced by the incorporation into a hypergraph, however it is still orders of magnitude bigger than written input. Moreover, the relation of edges can be assessed much more easily using a graphical interface than e.g. comparing vertex numbers of a chart.
- MILC uses complex data structures. The development of grammars and lexicons using text-based tools is usually fast and sufficient. The reasons for this are on one hand the property of many formalisms (including the formalism for MILC) to allow concise formulations using macros, and on the other hand the structures created (grammar rules and lexicon entries) are in many cases highly under-specified. However, if feature structures representing the final output of some processing stage are given as output, then a simple text-based representation is unsatisfying. A graphical presentation may use different fonts, sizes and a visual bracketing to aid the user in correlating different parts of a feature structure in a fast, efficient manner. The visualization implemented for MILC is no development tool for linguistic knowledge sources; there is no possibility of modification nor a mechanism to fold feature structures as a manipulation of the output form, as has been realized with the system Fegramed (Kiefer and Fettig, 1993).

We chose Tcl/Tk (Ousterhout, 1994) as implementation language for the visualization because of the efficiency (both in development and execution). That language forms the framework for the graphical interface for the visualization (a snapshot of the interface is shown in Figure 4.26). The figure shows a fragment of a hypergraph that is currenty being processed. The property of hyperedges having several start and end vertices is depicted by the ray-shaped connections of edges to vertices. One of the edges was selected using the mouse. The properties of that edge are displayed in the bottom half of the window.

These properties include the original input (“Herr Pfitzinger”), the information about the start and end vertices of the edge, and possible annotations from different components. These annotations are shown in the list on the left; one of them has been selected. The content of the selection is the target language surface representation (“Mr. Pfitzinger”) described by the feature structure to the right. The visualization of hyperedges and feature structures has not been implemented using Tcl/Tk directly; instead, they have been realized in an underlying C-layer.

The incremental visualization of word graphs in an appealing manner is astonishingly difficult (Florey, 1998). In particular, the incremental creation of a graphical representation by ensuring that certain layout characteristics are kept (e.g. the property that vertices are ordered by time), is difficult using conventional graph layout algorithms. The situation for MILC is in so far favorable, as we reduce the number of edges drastically by converting simple word graphs to hypergraphs. That

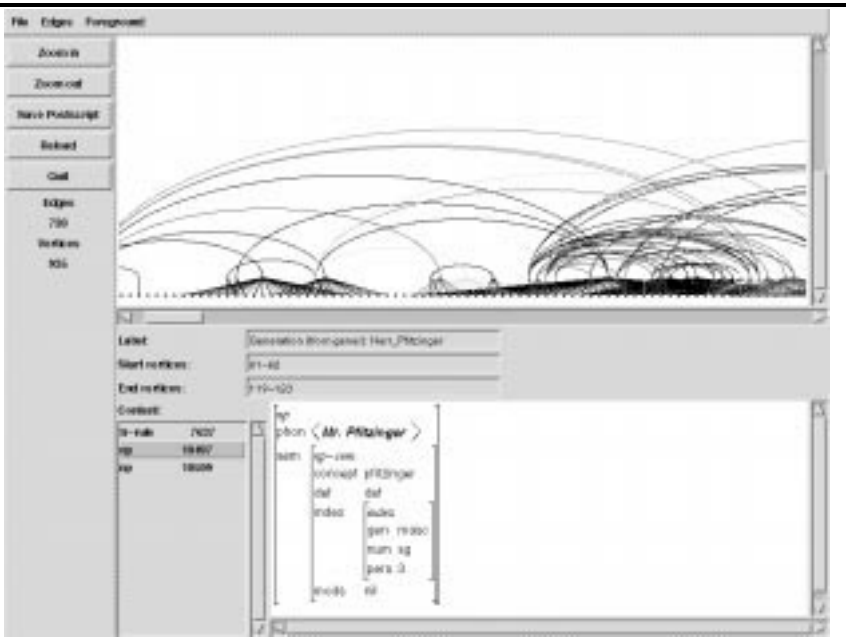


Figure 4.26. A snapshot of the processing with MILC

enables us to use a relatively simple drawing of vertices (linear and equidistant in the x-axis) and edges (as parts of ovals) without loosing too much clarity.

4.11 Extensions

MILC is a complete system for the translation of single, spontaneously spoken German utterances into English. We described the extent and the function of the system in the preceding sections. Despite the completeness, an augmentation or extension of the functionality is possible and desirable in several aspects. This is exactly within the intention of the implementation, which on one hand was to provide an overall incremental system for the translation of spoken language, offering on the other hand an experimental platform suitable for further investigations into the architecture of natural language processing systems in general and the role of incrementality and interactivity in particular. In this sense, MILC was designed to be an open system. In this section, we will point out several obvious modifications that can be implemented immediately. Moreover, we will present an extension consisting of two parts, covering the range of phenomena the system can handle within

an utterance as well as the extension to process more than one isolated utterance. Finally, we discuss some questions that arise in the context of an attempted anytime analysis.

The simpler extensions we want to mention are mostly local modifications of individual components that may have a direct impact on the performance or processing quality of a module. Examples for this type of modification are:

- **Language model**. The language model we use within MILC is a bigram model, predicting the probability of two words being adjacent. However, since the input utterances usually cover more than two words, the application of a trigram at least seems reasonable.
- **Extended idiom processing**. The treatment of idioms in the current version of MILC concentrates on routine formulas which are strictly lexicalized. They are not allowed to be inflected or otherwise structurally variable. In the domain of appointment scheduling this might be too restrictive, since there are many idiomatic expressions that show some degree of freedom in realization: For instance, proposals for an appointment are often realized idiomatically, but they frequently contain date expressions or interspersed modal particles, and thus cannot be adequately processed using a strictly lexicalized approach (Erbach and Krenn, 1994). The variable modeling of idioms necessary to do this can, for example, be achieved using finite state transducers that have access to the category information of words. Also, optional parts like particles can be captured like this (Zajac, 1998).
- **Grammars**. The rules of the grammars used in the system should be augmented by probabilities, following the method of Weber (1995). This approach of computing the score of a derivation, taking into account statistical information about the grammar itself, is not restricted to syntactic parsing. It can consequently be applied to transfer and generation as well.
- **Selection**. The evaluation function that searches for the best sequence of hypotheses in the generation graph can be revised. Two aspects offer themselves as being helpful:
  - The scoring function could additionally use an English language model to smooth the generation output.
  - The utterances that have been generated could be reanalyzed to a certain degree, thereby generating a better coherence between fragments of an utterance.
- **Generation grammar**. The rules of the generation grammar are ordered hierarchically. This results in a better control of their application and ensures that rules for specific cases are always treated with priority. Additionally, the performance of the generator can be improved by applying general rules only if no other more special rule can be used.
- **Chart structure**. At least the analysis algorithms currently demand the adjacency of words and constituents when integrating them into larger contexts. In Section 4.6.2, we examined the possibility of marking certain categories of edges that could be consumed without modification of the linguistic description. In Section 2.6, we presented the introduction of gaps within the structure of the chart.

A combination of both approaches could lead to a method capable of skipping enough material in the chart to construct a continuous analysis without introducing too much overhead.

- **Search strategy**. Left-connected graphs contain vertices that have no outgoing edges because no word hypotheses having a sufficiently high confidence could be found. The search strategy (again, at least within the analysis modules of the partial parser and the integrator) may be modified to take advantage of this fact. In order to do this, a schema of delayed evaluation could be introduced. Tasks for an edge are inserted into the global agenda only if the end vertex of the edge has at least one outgoing edge. Using this restriction, at least simple dead ends (those consisting of a single edge) are not considered. However, such a delayed evaluation interferes with the strict left-to-right paradigm, since we cannot predict how long to wait for the continuation.
- **Parallelization**. The structure and implementation of MILC guarantee a high degree of inter-modular parallelism of all components. We did not delve into the options of an intra-modular parallelization of individual components. It turns out that a concept using layered charts together with a shift-invariant formalism is a nearly ideal foundation for the parallel processing of graph-like structures. We have already shown in an earlier investigation (Amtrup, 1992; Amtrup, 1995c) that parallel parsing using a chart-based paradigm results in a potential performance increase. However, Amtrup (1995c) only covers the processing of written language. Spilker (1995) attempts to process spoken language with an agenda-based parallel system, but does not achieve satisfying performance on a shared memory multiprocessor system. Starting off from these two results, the distribution of parts of the analysis based on sub-paths of a word graph and using loosely coupled multiprocessors seems to be more appropriate. A variant that offers itself immediately is to take the edges emerging from each vertex as seeds of a distributed processing. Up to a certain degree, the graph is unfolded into a tree in order to utilize as many processors as possible. After sub-paths have been analyzed on different processors, the results have to be combined to reconstruct the original topology of the graph.
  The use of chart-based mechanisms throughout allows a system like MILC to extend parallel processing beyond the application to a syntactic parser. Other modules may also benefit from the availability of highly parallel computer systems, e.g. on the basis of loosely coupled transputer systems.

4.11.1 Extension of the Architecture

The extension we are going to present in this section touches two aspects of computation: First, additional properties of spoken languages are integrated into the processing. Here, we are mainly concerned with prosodic phenomena, to a lesser degree we are going to discuss the introduction of feedback loops that have not been implemented so far. Second, we extend the applicability of the system by taking into

account complete dialogs. This is done in order to be able to model effects that cover more than a single, isolated utterance.

Both directions of extension have been previously studied in different contexts, thus they present nothing new that would have been impossible without an architecture like MILC. Nevertheless, the investigation of both aspects and the exploration of their embedding into an integrated, uniform system seems to be desirable, e.g. in order to evaluate the mutual influence of design decisions within a complex system for processing of spontaneous speech. MILC is an ideal framework for this endeavor.

The influence of prosody has been extensively investigated in the Verbmobil project for the first time (Noeth *et al.*, 1997). Each word hypotheses is annotated with probabilities that indicate if the hypothesis carries a phrasal accent, if it marks a phrase or sentence boundary and if it marks a certain sentence mood. This information is used by the following modules for linguistic analysis to modify the scores of rules and thus increase the efficiency and accuracy of the analysis itself (cf. Amtrup and Jekat, 1995). With the aid of prosodic evidence, the syntactic analysis of the Verbmobil project could be augmented considerably, reducing the number of syntactic readings by 90%. A different aspect of prosody was used in the architectonic prototype INTARC. Here, prosodic information was used to calculate the focus of an utterance, which in turn was used to produce a shallow translation based on dialog acts (cf. Jekat, 1997). An analysis of the underlying corpus showed that focused words are often used as the basis of determining the dialog act conveyed in the utterance. Thus, a prosodically motivated keyword search predicts the dialog act (Elsner and Klein, 1996).

The integration of prosodic evidence can in principle be achieved in two different ways. The method accepted for Verbmobil consists of annotating recognized words with prosodic information after the initial word recognition. This is partly due to the fact that the syllabic structure of a word has to be known in order to compute its prosodic feature vectors. Such an additional annotation can be integrated seamlessly into the architecture of MILC. The only possible obstacle would be contradicting prosodic information of two word hypotheses that are to be integrated into a single hyperedge. In this case, the simple solution would be to create a new hyperedge. However, there are approaches that do not depend on previously computed word hypotheses. These methods use only content of the signal and do not rely on a word segmentation (Strom and Widera, 1996). The results of this method are quite good (the recognition rate for phrase boundaries is approximately 86%) and can be integrated into the layered chart as additional edges that may be used by any component.

The second augmenting measure that can be taken within the context of a single utterance is the introduction of feedback loops between components. In particular, this seems promising at the crucial speech-language interface, between components for a linguistic analysis and for speech recognition. Such a mechanism has been used within INTARC to increase the efficiency of the system as a whole. As already mentioned in the introduction, the feedback loop was established between the word

recognition and the syntactic parser (Hauenstein and Weber, 1994). MILC does currently not use feedback loops, but the design and implementation are specifically aligned with their exploration. Besides the interweaving of recognition and syntax the employment of influences from higher linguistic levels is reasonable and desirable (Menzel, 1994).

These higher linguistic levels come into view as we discuss the second extension of MILC, namely the processing of complete dialogs instead of single utterances. So far, MILC operates on single utterances of participants in a dialog without storing and using the information that was gathered during processing of an utterance. To do this is advantageous from several points of view. First, it is obvious that the coherence of a text is established over several utterances in a dialog. The information extracted from previous sentences can be used to disambiguate the content and communicative goals of the current utterance. Some utterances are only understandable by taking into account context and can only be translated in an appropriate way using that context (cf. Alexandersson, Reithinger and Maier, 1997). Thus, the incorporation of a dialog management component into MILC is a suitable goal. We can leave it an open issue for now whether the dialog component stores the knowledge about the utterances internally, or whether the structure of the layered chart is extended in order to capture the complete dialog. In order to carry out disambiguation, the integration of several utterances into the chart is not necessary, this function may be fulfilled on a question-answer basis as in Verbmobil (cf. Alexandersson, Maier and Reithinger, 1995). If, however, the dialog processing is to influence other components by means of a feedback loop, then an extension of the chart seems preferable for reasons of uniformity of information storage. This schema requires a slight modification of the chart structure in order to be able to remove all utterance internal information once an utterance has been completely processed.

The extension of the architecture of MILC in the direction of a dialog processing machine is also favorable from the viewpoint of generation. Only if a system has access to translations already carried out and verbalized, a strategic or tactical text planning becomes feasible, e.g. to augment the style of utterances by replacing definite descriptions by pronouns for their second occurrence.

4.11.2 Anytime Translation

Anytime algorithms have been employed mainly for planning problems so far (Russel and Zilberstein, 1991; Boddy and Dean, 1994). The application of anytime procedures is highly desirable for natural language processing, as we already mentioned in the introduction. However, there are many problems of theoretical and practical nature with their use in highly modular, natural language processing systems that work with complex feature formalisms. First, we have to define what the impact of anytime algorithms would be for the function of a system, and what kind of effect users may realistically expect. Only after this, the consequences for dependencies between single algorithms and modules can be predicted in a satisfying manner.

The time profile of a system processing spoken language can be divided into two sections, if we assume the application of interpreting within a dialog:

- The first phase is characterized by the time that a speaker needs to finish his or her dialog contribution. Depending on the situation and the speaker, the duration of this phase is between a few seconds and one minute.[32]
- The second phase consists of the period that the user has to wait for the translation of the machine. Once the speaker has completed an utterance, his attention span is relatively short and the relevance of a translation degrades fast.[33]

Russel and Zilberstein (1991) distinguish between two different kinds of anytime algorithms. *Interrupt algorithms* may be interrupted at any point during their computation and are always able to deliver some results. *Contract algorithms* on the other hand, have been informed beforehand about the amount of time they may spend for an analysis. If they are interrupted before this time span has ended, they produce no output. After the computing interval ends, they are able to deliver results. The common property of both variants is that the quality of results increases monotonically with the time the algorithms spend for the computation. This is modeled using a probabilistic performance profile.

It is apparent from this definition that contract algorithms are primarily not applicable in the framework of natural language processing we assume here. The length of the first phase mentioned above is unknown in advance, the second phase is very short. The first point in time at which it would be possible to get an estimate about the remaining time is the borderline between those two phases. In principle, interrupt algorithms seem to be the method of choice.[34] Now, the question would be, what the granularity of such an algorithm is, i.e. in what intervals it is reasonable to extract a result. Essentially, this is the question of establishing a quantization of time. Görz and Kesseler (1994) investigate this issue for the structural analysis within the INTARC system and report intervals of 10 or 100 ms. Individual unifications should be atomic in their opinion, since otherwise there might be unsolved coreferences.[35] Within a chart-based system, two choices of intervals offer themselves: The execution of one individual task on the agenda, or the execution of all tasks at one point in time.

---

[32] Utterances lasting longer than one minute are rare and in the framework of appointment scheduling almost always artificially elicited. However, cultural influences play a certain role for the length of utterances.

[33] An interesting method to prolong the time that a system may use for the processing of input, is to "mumble" about the current state of the analysis during the computation (Karlgren, 1994).

[34] Russel and Zilberstein (1991) show that every contract algorithm can be converted into an interrupt algorithm. The algorithm thus constructed needs three times longer to obtain results of equal quality.

[35] However, they claim that it would be possible to define core features and features of lesser importance within a type lattice, thereby allowing for a phased unification.

Further consideration reveals that it is not useful to interrupt a system processing spoken language before the input utterance is completely spoken. This may be trivial, but it effects the measurement of time in the system. Previous work within the anytime framework for planning assumes that the input is initially available, providing the data for the computation of the result. The input is atomic and conceptually does not consume any time. However, under the circumstances present here, the construction of the initial condition is by no means instantaneous. To the contrary, the time needed for input amounts to the majority of processing time available. Thus, the total computation time is the sum of the two phases mentioned above; an interruption during the first phase is, however, impossible. After completion of the first phase, the system may be interrupted in certain intervals (e.g. 100ms), and delivers a result with monotonically increasing quality.

The issue of the quality of a result is widely undecided. In the area of machine translation, th structural complexity of a translation (or, better, the complexity of the tree representing the translation) may be a first clue, but without having the status of universal validity. For that reason, artificial methods for the processing of natural language can only be classified as *weak anytime* algorithms (Menzel, 1994), having no general performance profile, but only a profile based on the actual input. In the MILC system, we provide continually better results by applying the selection function within the generator component. In this respect, the incremental output of utterance fragments can be viewed as a first step towards the preparation of output with a monotonically growing quality.

Of course, MILC does not implement an anytime behavior. While being incremental throughout, it is not reasonable to interrupt the execution before generation hypotheses for a large part of the input has been computed. Thus, MILC cannot react to time constraints below a certain threshold.

The incremental operation of all modules in a system, on the other hand, is a necessary precondition for the introduction of anytime behavior. Assume a non-monolithical system. If such a system works non-incrementally, and if that system is interrupted, there will be situations in which only a subset of the modules has completed their tasks. Thus, no result can be presented in response to the interrupt. Even if contract algorithms are used, the resources that one component needs to complete its operation depend on the output of preceding components and can thus not be easily determined.

Incremental modular systems are in principle able to cope with anytime demands. If one assumes that the increments of individual components are small enough, a repetitive operation and increase of quality is possible. The strategy would be to produce a very weak output first (e.g. a word-for-word translation). Further iterations of processing would analyze the input utterance again and try to increase the quality of results. A selection function evaluates the set of results obtained so far, and is able to improve them. Such a behavior, as is exemplified by MILC, avoids the commitment for intermediate results and leaves the ultimate decision for results open as long as possible. This is done at the cost of inducing an increased effort to process a high number of alternatives. Besides the definition of

quality, which we already discussed, the question of how much effort has to be invested in the augmentation of an insufficient result is also of great importance, i.e. a dynamic cost-effectiveness analysis. Here, no definite solutions exist either, only heuristics can be thought of, the impact of which have not yet been investigated. A first approach could be to include constraint satisfaction algorithms (Menzel, 1998).

The procedure of translation in the framework of anytime algorithms can be more precisely stated:

- While the user is speaking and the input is not yet complete, each component tries to process tasks as far to the right as possible. This corresponds to working as time-synchronously as possible. A component may produce additional results only if the recognition process leads to delays in the production of word hypotheses.
- At the end of the utterance[36] this behavior changes. For a short period of time the time-synchronous processing has to continue, up to a point when all components have initially processed the whole input. At this point, a raw translation is complete. Therefore, the augmentation of the translation quality is the foremost goal now. A reasonable strategy would first identify segments of the input that resulted in a translation of poor quality and invest computing time in order to come up with better results. The iterative processing and consideration of these intervals may achieve the biggest increase in overall translation quality, it is thus worth processing them with priority.

4.12 System Size

MILC, as well as some of the support programs (grammar compiler etc.), are completely implemented using C++. For the implementation of the visualization we used the scripting language Tcl/Tk (Ousterhout, 1994), which has been extended by C-routines to draw edges and feature structures. Table 4.5 shows the current size of the MILC system in lines of code.

4.13 Summary

In this chapter we explained the implementation of MILC. We discussed the architectonic foundation, the behavior of each component of the system and their cooperation within a distributed system. The predominant motivation was to implement

[36] Even the recognition of the end of an utterance is a non-trivial problem. If one assumes that a period of silence marks the end of a user's input, then it is disputable how long the silence has to be in order to unambiguously mark the end. In the first phase of Verbmobil, the user had to push a button while speaking; thus, the detection of the end of an utterance was simple.

Table 4.5. System size (in lines of code)

Component	No. of lines
Communication and configuration	
Communication system ICE	10715
Configuration	1158
Misc.	
Trees and lists	1794
System statistics, object creation	465
Graph analysis	
DAGs	4825
Analysis system	360
Feature structures	
Core	6178
Functions	103
Chart	
Chart maintenance	1428
Agenda maintenance	244
Knowledge sources	
Grammar maintenance	454
Lexicon maintenance	317
Compiler for grammars and lexicons	240
Type lattice	464
Grammars	2494
Lexicons	4322
Components	
Recognizer	362
Idioms processor	697
Parser	825
Integrator	765
Transfer	779
Generation	439
Visualization	1118
Total	40546

a complete system translating spontaneous spoken language, which follows the incremental paradigm. Additionally, we wanted to provide an architectonic framework that could be used for future research and experiments. This motivation first lead to the design of layered charts, a data structure supporting the development of highly complex, distributed software systems for natural language processing. The integration that is achieved by a layered chart is complemented by the use of an uniform formalism within the system. Layered charts have properties which make them superior to data structures employing a central control (like blackboards and whiteboards).

The validity and suitability of the data structure thus implemented have been confirmed by the complete implementation of the translation system MILC. A comprehensive implementation ensures at the same time that a theoretical advantage can be realized in practice. A partial implementation in whatever way could not have been accepted as a proof of this. All components (word recognition, idioms processing, partial parsing, utterance integration, transfer and generation) are entirely incremental. This enables the system to preserve this type of operation until English surface fragments are generated. MILC works on the level of single utterances.

The main desirable extensions in a conventional framework are the utilization of prosodic information for disambiguation and the incorporation of whole dialogs. As for architectonic aspects, the introduction of feedback loops seems promising, especially at the crucial speech-language interface.

Chapter 5

Experiments and Results

In this chapter we describe the experiments that have been carried out with the MILC system. The most important findings are given by the investigation of the translation of several dialogs from the Verbmobil domain. Accompanying experiments deal with the usability of hypergraphs for the underlying task and an attempt to estimate how efficient incremental systems may be in comparison with conventional approaches.

In the preceding chapters, we presented the theoretical foundation, the architecture and implementation of the MILC system. We argued that having an experimental work bench for experiments in the framework of incremental translation was one of the primary motivations to implement the application. To show the adequacy of incremental methods leads to the complete implementation of the system, as a partial implementation could not have been used to validate this. In order to further demonstrate that such an incremental system is actually capable of coping with real-world input, we investigate dialogs in this chapter that have been recorded for the Verbmobil project.

First, we will demonstrate the effect that using hypergraphs has on the performance of a parsing system. The result is that the transition from conventional graphs to hypergraphs having edges with sets of start and end vertices facilitates the analysis of large, incremental word graphs and makes it possible to achieve processing times that are in acceptable dimensions. Then, we will investigate complete dialogs in the appointment scheduling domain, the translations of which were evaluated by native English speakers. The last experimental section in this chapter deals with the question of whether the use of a multiprocessor system actually speeds up the computation. We also investigate the relation between an incremental parallel implementation and a comparable non-incremental implementation on a single processor.

All experiments we describe in this chapter (with the exception of the hypergraph conversion) have been executed an a Sun UltraSparc 4 with two processors and 1 GB of main memory. The operating system used was Solaris 2.6, the experiments were carried out under low workload (load values of 0.03 without MILC).

J.W. Amtrup: Incremental Speech Translation, LNAI 1735, pp. 155-168, 1999.

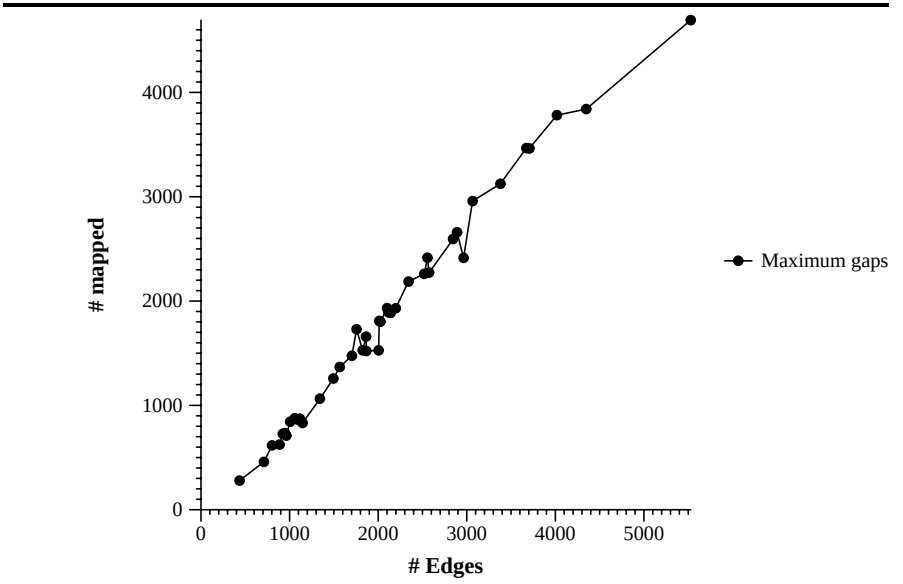


Figure 5.1. Reduction of word edges by hypergraph conversion

5.1 Hypergraphs

The main modification that distinguishes hypergraphs from ordinary word graphs for the processing of natural language is their ability to allow multiple start and end vertices for a single edge. It follows that the number of vertices will not change, but the number of edges should decrease drastically by converting word graphs into hypergraphs. This should have severe consequences for the runtime of modules for processing spoken language.

In order to evaluate both phenomena, we investigated a dialog in the domain of appointment scheduling. We carried out experiments with the dialog n002k from the Verbmobil corpus. The dialog contains 41 turns with an average length of 4.65 seconds of speech. The word graphs produced by the Hamburg word recognizer contain 1,828 edges on the average.

Figure 5.1 shows the reduction in the number of edges by transforming the word graph into a hypergraph. Only 157 edges remained on the average, a reduction of approximately 91%.

In order to assess the consequences of such a modification for the performance of a linguistic analysis, both the word graphs and hypergraphs have been subject to partial parsing. We used the knowledge sources already mentioned in Section 4.6,

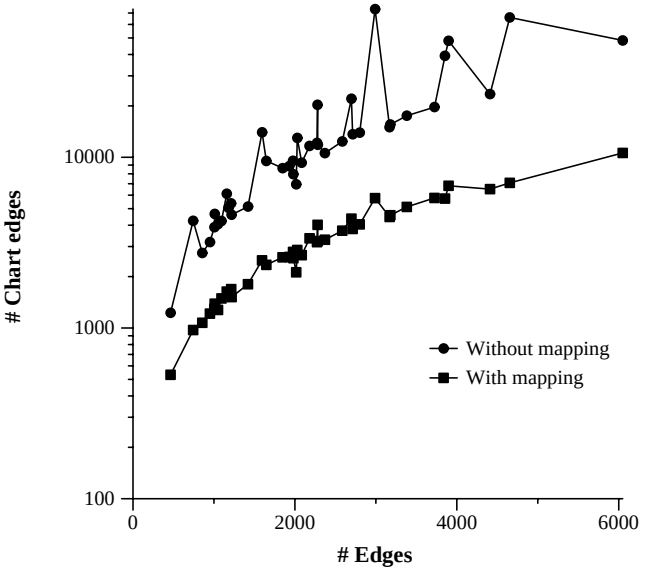


Figure 5.2. Reduction of chart edges by hypergraph conversion

i.e. a type hierarchy with 345 types, a lexicon with 413 entries and a grammar with 80 rules.

We use two parameters to characterize the performance gain achieved by using hypergraphs: The number of edges in the chart that are created during the course of parsing, and the total time used to process the input. The performance increase we saw is within the same order of magnitude as the reduction in the number of edges in the input graph; the gain is not exponential, as one may have expected if all possible paths would have been considered. The main reason for this is the redundancy test that is carried out inside the parser. The parsing of the original word graphs created 15,547 edges on the average, while only 3,316 edges were created for hypergraphs, a reduction of 79%. The detailed analysis is shown in Figure 5.2. The reduction in parsing time is approximately identical, as Figure 5.3 shows. Parsing of word graphs took 87.7 seconds, parsing of hypergraphs 6.4 seconds, a reduction of 93%. The number of spurious analyses resulting from hypergraph conversion (cf. Section 2.6.3) is negligible. The analysis of the original dialog generates 5,586 syntactic constituents, while the analysis of hypergraphs creates 12 additional constituents, an over-generation of 0.2%. Moreover, the additional hypotheses are very short (two or three words).

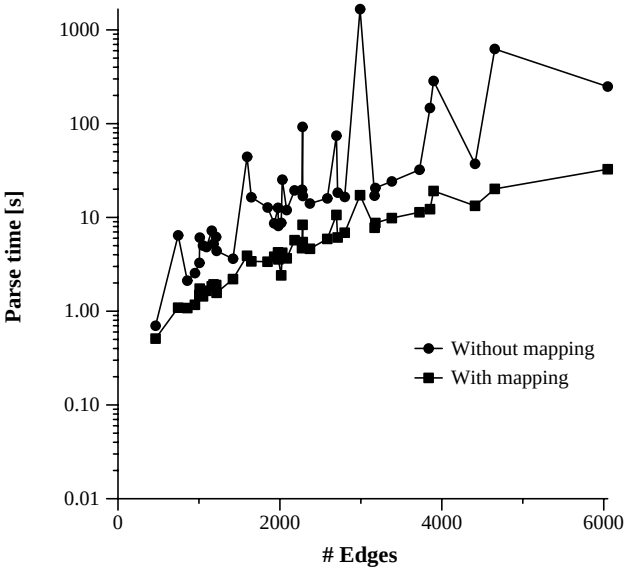


Figure 5.3. Reduction of analysis time by hypergraph conversion

However, there were extreme cases in which the hypergraph analysis was 94 times as fast as a word graph analysis. One turn had to be excluded from the test set, because it could not be processed as a word graph due to lack of system memory. The experiments mentioned here were conducted on a Sun Ultra SparcStation 1 with 128 MB of main memory.

5.2 Translation

5.2.1 Data Material

The recordings, which are the used in the investigation reported here, contain five dialogs from the Verbmobil domain. These are dialogs whose goal is the scheduling of an appointment. Some characteristic properties of the dialogs are shown in Table 5.1. We use data that has been used for the acoustic evaluation and that thus have not been used for the training of the word recognizer we used (Huebener, Jost and Heine, 1996). The dialogs were spoken by people of different gender and the word accuracy is around 70%. This result was achieved by using three active tokens per

Table 5.1. Properties of the dialogs used in the experiments

Property	Dialog j512	Dialog j533	Dialog j534	Dialog m111	Dialog m123
No. utterances [#]	17	6	24	24	11
Length [s] (min)	0.76	0.65	0.83	0.67	0.96
(max)	36.09	18.89	17.29	10.19	10.7
(∅)	8.50	7.84	7.44	3.16	5.25
No. words (∅)	23.8	24.3	22.2	10.5	14.9
speakers	m/m	m/f	m/m	f/m	f/m
word accuracy (%)	64.8	68.1	66.2	70.3	76.8
word accuracy best path (%)	43.6	37.0	42.6	40.2	58.3
No. vertices (∅)	784	699	687	302	470
No. edges (∅)	7340	7904	6869	3828	4342
No. paths (∅)	$7.5 \cdot 10^{213}$	$7.4 \cdot 10^{150}$	$8.8 \cdot 10^{106}$	$8.3 \cdot 10^{81}$	$3.7 \cdot 10^{63}$

state of the word recognition net describing the recognizer lexicon. The threshold for the evaluation of partial results within words and at the end of a word was 100.

The word accuracy we use here is several percentage points below the current best values for the data involved (Lehning, 1996). However, the evaluation data is usually computed non-incrementally and undergoes severe modifications, which usually are not useful for incremental processing (Amtrup, Heine and Jost, 1997; Jost, 1997). Moreover, the investigation into graphs with such a low recognition rate is in principle what we want to do, since they correspond to the current best results obtained for speech much less restricted than in Verbmobil. Currently, they are in-between the *Switchboard*-corpus and the *CallHome*-corpus (Finke *et al.*, 1997).

Table 5.2 gives an impression of the type of dialogs used. It shows the transliteration of utterances in the dialog m123. The extralinguistic hypotheses used are ⟨NIB⟩ (non-interpretable area) and ⟨SPELL⟩ (spelling entities, here the branch O.B.K.I.). We show a literal translation for each of the turns.

The five dialogs have been separated to give a qualitative assessment of the coverage of the knowledge sources integrated into the system. The dialogs m111 and m123 were used to develop the grammar and lexicon, while the dialogs j512, j533 and j534 have been analyzed without any modification of the grammar. The lexicon was augmented to cover the words occurring in the new dialogs.

5.2.2 Linguistic Knowledge Sources

All linguistic modules use a common type lattice in order to guarantee the compatibility of feature structures in the system. This type hierarchy consists of 633 types. The structure of the lattice has already been described in Section 3.3.1. We

Table 5.2. The utterances in dialog m123

Turn	Utterance
m123n000	guten Tag Herr ⟨NIB⟩ Klitscher hier ist wieder Fringes ich möchte gerne diesmal einen Termin ⟨NIB⟩ für das Arbeitstreffen in der Filiale ⟨SPELL⟩ in Potsdam mit Ihnen vereinbaren ⟨NIB⟩ *good day Mr. Klitscher hier is again Fringes I would like this time an appointment for the work meeting in the branch ... in potsdam with you schedule*
m123n001	⟨NIB⟩ guten Tag Frau Fringes ja wie sieht es denn bei Ihnen aus in der Woche vom sechsten bis zehnten Mai *god day Mrs. Fringes yes how looks it then with you out in the week from sixth to tenth may*
m123n002	⟨NIB⟩ sechster bis zehnter Mai ach da muß ich unbedingt zu einem Seminar nach Rothenburg ⟨NIB⟩ *sixth to tenth may well there must I really to a seminar to Rothenburg*
m123n003	⟨NIB⟩ dann schlagen Sie doch einen Termin vor der Ihnen passen würde *then propose you then a date on which you fit would*
m123n004	ja also wie wäre es denn im April vom zweiundzwanzigsten bis zum sechsundzwanzigsten *yes well how would it then in april from twentysecond up to twentysixth*
m123n005	oh das geht ⟨NIB⟩ leider nicht da habe ich ein Seminar in Koblenz ⟨NIB⟩ haben Sie noch einen anderen Termin *oh that goes regrettably not there have I a seminar in Koblenz have you another a different date*
m123n006	ja lassen Sie mich mal sehen ⟨NIB⟩ ja da hätte ich noch im Juni vom zehnten bis zum vierzehnten einen Termin frei ⟨NIB⟩ *yes let you me then see yes there would I still in june from tenth to fourteenth a date free*
m123n007	das würde mir auch passen ja sollen wir uns dann in Potsdam treffen in der ⟨NIB⟩ Filiale ⟨SPELL⟩ *that would me also fit yes shall we us then in Potsdam meet in the branch ...*
m123n008	ja machen wir das *yes make we that*
m123n009	okay auf Wiedersehen Frau Fringes ⟨NIB⟩ *OK good bye Mrs. Fringes*
m123n010	auf Wiederhören *good bye*

used the lattice to construct the relevant knowledge sources, among them four lexicons (idioms, analysis, transfer and generation) and four grammars (partial parsing, utterance integration, transfer and generation):

- The lexicon for idiom recognition consists of 21 entries for fixed expressions that cannot otherwise be analyzed compositionally, like "guten Tag" (how do you do) or interjections like "einen Moment" (just a second). Additionally, we used the idiom recognition for the modeling of multi-word prepositions (e.g. "bis auf",

expect from). Using this method, the description of prepositional phrases in the analysis grammar may assume that prepositions are single words, the construction is left for idiom processing. Naturally, the idiom recognition does not use a grammar.

- The partial parsing stage uses two sources of linguistic knowledge: A phrase structure grammar and a lexicon. The grammar consists of 35 rules for the construction of "small" phrases. The attachment of complements and adjuncts has not been modeled. The set of rules can be divided into rules for describing date expressions, prepositional phrases, noun phrases and the construction of modifying adverbial phrases. Adjectives are integrated into the noun phrases that dominate them.
  The corresponding lexicon consists of 772 entries. We use a full form dictionary with a lexical ambiguity of 1.15, i.e. there are 1.15 feature structures describing different facets of a surface word. If restricted to verbs, the lexical ambiguity is 1.38. The lexicon contains entries for verbs, nouns, adjectives, adverbs, determiners, date expressions, conjunctions and interjections.
- The utterance integration does not need a lexicon. All relevant structures have already been created by the partial parsing stage, in particular, we carry out lexical access for verbs and distribute the results as preterminal edges to the integration. The grammar of the integration phase foremost models the binding of complements and the attachment of prepositional and adverbial phrases. The grammar contains 18 general rules.
- For transfer we need a grammar, as well as a lexicon. The grammar describes the transformation of German semantic descriptions into their English counterparts. It contains only 10 rules, most of which deal with the transformation of verbal and nominal expressions. Part of the rules describe the transfer of modifiers. Date expressions, however, which account for a lot of rules during analysis, are simply takes as they are. The lexicon used in this part of the system captures the mapping of semantic concepts and is mainly ordered by word classes. Currently, it contains 207 descriptions.
- The generation produces English surface representations from English semantic descriptions. The lexicon used for this purpose contains 311 entries, noticeably less that the lexicon used for analysis. Besides the easier English morphology, the reason for this is that we did not assign a high importance to stylistic issues during generation, but instead maintained a correct and adequate realization of the content. The grammar contains 40 rules and describes the necessary syntactic concepts like verbal phrases, noun phrases and preposition phrases, adjectival and adverbial modifiers.

5.2.3 Experiments and System Parameters

The dialogs described earlier have been analyzed using the MILC system. We recorded runtime and the output of the generation stage, as well as some interesting system parameters. Table 5.3 shows the relevant results. All computations

Table 5.3. Results of analyzing five dialogs

Parameter	Dialog j512	Dialog j533	Dialog j534	Dialog m111	Dialog m123
Runtime [ms]	63977	48974	54725	17361	19352
No. translations [#]	51	45	44	24	28
No. edges [#]	9876	11065	8849	4952	5740
No. tasks [#]	92579	190212	93961	44013	37793
... pruned [#]	48137	137019	47121	21764	13600
Main memory [MB]	81	102	92	48	55
Unifications [#/s]	2068	3048	4520	3018	3622

have been carried out with a language model weight of 5.0 and a beam search which allowed for 200 tasks per vertex, cutting off all others. All values represent average values for the processing of one utterance within a dialog. The runtime is measured as wall clock time from the loading of the first component up to termination of the system.

In order to give an impression of the type of output, we show the sequence of generator results for the utterance j534a005 ("⟨NIB⟩ ⟨NIB⟩ oh das ist schade da bin ich zu einem geschäftlichen ⟨NIB⟩ Termin in Freiburg aber wie ist es denn vom vierzehnten bis zum neunzehnten") in Table 5.4. It is plain to see how recognition errors lead to misinterpretations and incorrect translations ("Termin in Freiburg" comes out as "appointment on Friday"); moreover, the incremental nature of MILC can be seen in the growing length of output hypotheses from the generator. It searches continuously for the best path through the current hypergraph of English fragments and issues it. The processing of the turn shown here led to 130 English utterance fragments. These fragments constitute the graph in which surface hypotheses are searched.

5.2.4 Evaluation

We presented the translation created by the system to native English speakers to evaluate the results achieved by MILC. After a short introduction into the dialog setting and the special kind of results provided by MILC (i.e. as a sequence of fragments in English for each utterance), the subjects recorded three pieces of information for each turn. They were asked to note the most important pragmatic content of each utterance, which was chosen from a list of dialog-act similar classes, namely Greeting, Goodbye, Request for a proposal, Proposal, Denial, Reason, Acceptance.[1] Multiple choices were possible. Moreover, the subjects could issue a date relevant

[1] This list is a selection from the dialog acts used in Verbmobil, cf. Jekat *et al.* (1995).

Table 5.4. Generator output for the utterance j534a005

Yes
Yes |let's say
my |let's say
me |Yes |let's say
me |Yes |it |let's say
me |Yes |it |for me |let's say
me |Yes |it |for me |let's say |my
me |Yes |it |I would suggest
me |Yes |it |I would suggest |you
me |Yes |it |I would suggest
me |Yes |it |I would suggest |an appointment
me |Yes |it |I would suggest |an appointment |us
me |Yes |it |I would suggest |an appointment |we
me |Yes |it |I would suggest |an appointment |friday
me |Yes |it |I would suggest |an appointment |friday |we
me |Yes |it |I would suggest |an appointment |friday |we |us
me |Yes |it |I would suggest |an appointment |on friday |we
me |Yes |it |I would suggest |an appointment |on friday |we |us
me |Yes |it |I would suggest |an appointment |on friday |we
me |Yes |it |I would suggest |an appointment |we
me |Yes |it |I would suggest |an appointment |we |us
me |Yes |it |I would suggest |an appointment |we
me |Yes |it |I would suggest |an appointment |we |you
me |Yes |it |I would suggest |an appointment |we |something
me |Yes |it |I would suggest |an appointment |we |the fifth
me |Yes |it |I would suggest |an appointment on friday |we |the fifth
me |Yes |it |I would suggest |an appointment |we |the fifth
me |Yes |it |I would suggest |an appointment on friday |we |the fifth
me |Yes |it |I would suggest |an appointment on friday |we |something |the fourteenth
me |Yes |it |I would suggest |an appointment on friday |we |from the fourteenth
me |Yes |it |I would suggest |an appointment on friday |we |suggest maybe |you |
me |Yes |it |I would suggest |an appointment on friday |we |suggest maybe |you |the fourteenth
me |Yes |it |I would suggest |an appointment on friday |we |suggest maybe |you |the fourteenth |up to the nineteenth

Table 5.5. Evaluation of the translations

Parameter	Dialog j512	Dialog j533	Dialog j534	Dialog m111	Dialog m123
Pragmatics correct [%]	74	65	58	63	81
Date correct [%]	61	45	46	66	45
Quality [1–4]	2.3	2.1	2.4	2.3	1.9

for the utterance, if possible. Finally, we asked for the subjective quality of the translation, which could be chosen from four values (Good, Rough, Understandable, Not Intelligible).

Table 5.5 shows the result of this evaluation. The information produced by the subjects was compared to values obtained from the original German utterances. We show the percentages of correct answers for dialog act and central date. The quality measures have been mapped to numerical values, 1 meaning good and 4 meaning not intelligible.

This kind of evaluation is relatively coarse. Neither can the influence of an individual component be singled out, nor can definite trends be identified, as the data basis is too small. In order to perform such an evaluation, more restrictive criteria would have to be developed for the evaluation itself, bigger test corpora would have to be analyzed, and the number of subjects performing the evaluation would have to be higher. However, such a rigid evaluation cannot be carried out in the framework presented here, this is the domain of larger projects (Jost, 1997; Carter et al., 1997). Nevertheless, the evaluation we conducted here is suitable to assess the qualitative power of a system like MILC.

5.2.5 Extensions

The results of the evaluation show that a large portion of the input for MILC can be processed adequately. The success rate can be given as 60.4% approximately correct translations (Wahlster, 1997). The runtime of the system is about six times real-time. If we assume that the processing starts immediately upon the start of the speaker's utterance[2], then the user has to wait for approximately 34 more seconds after finishing his utterance for the complete translation. This value is relatively high — it is certainly beyond the patience of a casual user — but nevertheless in a range which allows further experiments.

The dialogs j512, j533 and j534 have not been used to augment the grammar. An analysis shows that the processing of subordinate sentences (e.g. j533a000: "...

[2] We do not take into account the time needed for recording and word recognition.

Table 5.6. Comparison of runtime for incremental and non-incremental configurations

Configuration			Dialog j512	Dialog j533	Dialog j534	Dialog m111	Dialog m123	Rel. runtime
I	I	-	63977	48974	54725	17361	19352	1.17
I	N	-	113245	99752	86179	29541	33457	2.08
N	I	-	22012	26692	40101	13915	14070	0.67
N	N	-	41322	39490	53600	19380	20634	1.00
N	N	B	56017	57486	73310	27377	28489	1.39

nachdem der letzte Termin im Juni gewesen ist würde es mir im Juli nicht so ⟨UNK⟩ gut passen ...") and assertions with a spontaneous word order (z.B. j533a001: "bin ich sehr mit einverstanden") have not yet been fully incorporated. Thus, a more elaborate analysis of structural variants and their incorporation into the grammars is necessary. Moreover, a better model for tempus and conjunctive expressions (needed mainly for politeness reasons) seems adequate.

As a concluding note we would like to add that an investigation like the one presented here cannot reach a coverage like bigger projects, e.g. Verbmobil (Wahlster, 1997) or Janus (Lavie *et al.*, 1997). In our opinion, however, the modeling of a restricted aspect of a domain like appointment scheduling is a test case well suited for the evaluation of the power of new architecture schemata. The effort for an analysis (and, thus, the runtime) grows more than linear with the number of lexicon entries, especially if the ambiguity increases due to multiple categories for individual words. Also, the addition of grammar rules leads to an increase in effort. However, the same means lead to a more selective application of rules. We think that both effects are approximately balanced. Nevertheless, one of the modifications we mentioned in Section 4.11, the augmentation of rules with probabilistic scores, should be implemented, as this would reduce search times.

5.3 Comparison With Non-Incremental Methods

An interesting aspect in investigating incremental techniques is the question as to how far the inevitable increase in processing effort can be made up by employing inter-modular parallelism. In order to approach this issue, we conducted a number of experiments and measured runtimes for different system configurations. The results are shown in Table 5.6.

The dialogs in the test corpus have been analyzed repeatedly, we measured total system running time. The first column describes the configuration of the system

we used for each measurement. The first letter denotes whether incremental, left-connected word graphs have been used as input (I) or if we transformed them to conventional word graphs beforehand (N). The second letter characterizes the processing strategy of MILC. If this letter is I, then we used the normal configuration, only data dependencies led to delays in the processing. If the letter is N, then we artificially serialized the processing. In order to do this, we introduced a protocol that allows a component to start only if the preceding components have finished their work. This is possible without any immediate problems, as MILC does not use feedback channels in the current version. However, we have to note that the serialization is not complete. On one hand, the start of each component is carried out in parallel on the multiprocessor we used, and on the other hand the underlying communication software works concurrently with other processes. Therefore, we introduced a further configuration that is characterized by the third letter. If we marked a B here, we started one additional process in order to create a high load for one of the processors. It carries out simple arithmetic calculations in an infinite loop. This means that we can assume MILC as being restricted to one processor with a sufficient amount of precision, while all other parameters, like processor type, memory space etc., can be kept stable.

The results of the experiments show that incremental approaches can well compete with non-incremental methods. As expected, the processing of non-incremental graphs (connected graphs that do not contain dead ends) is consistently faster than the processing of corresponding incremental graphs. It is not astonishing that the incremental processing was always faster than a non-incremental processing, since parallel processing could be employed. Instead, the main result that can be drawn from the experiments is that the incremental processing of incremental graphs is only 17% slower than the non-incremental processing of non-incremental word graphs. Until now, the processing of left-connected graphs was always much less efficient than the analysis of conventional graphs, even though parallelism could have been used. Thus, so far incremental approaches were unfavorable. The results we have shown here suggest that incremental approaches may show an efficiency similar to conventional algorithms by exploiting means we have described in this monograph. An absolute necessity for this kind of results is, of course, the use of parallel computers. This is especially true, since the machine we used only possessed two processors, while there are at least four high-load processes in the MILC system (Parsing, Integration, Transfer, Generation).

However, the statements made are only valid for the narrow case we described here. The non-incremental version of the processing modules was created by artificially serializing incremental algorithms. There is no doubt that a careful conventional implementation would have used different processing strategies in several places. As it is, the linguistic modules still operate from left to right in time. All possibilities for optimization, which arise from the accessibility of the complete input signal, have not been used. Thus, a strictly correct comparison between incremental and non-incremental systems can only be carried out if the non-incremental version

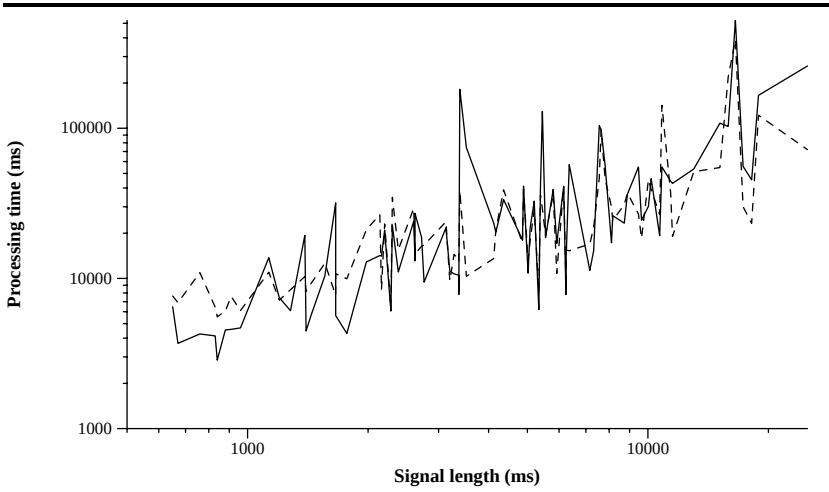


Figure 5.4. Comparison of incremental (—) and non-incremental (- - -) processing

is similarly optimized. We estimate that optimization could accelerate a system by a factor of two or three.

Figure 5.4 gives a detailed view on the processing times for different utterances. It shows the processing time for incremental (continuous line) and non-incremental (dashed line) system runs over the length of a single utterance. The trend that can be seen is that the employment of incremental algorithms has a stronger effect on shorter utterances as compared to relatively long turns. This may be due to a unproportional big unfolding of left-connected word graphs for long turns. A possible remedy for this effect involves the introduction of a delayed evaluation, which only processes an edge if there is a possibility of extending an end vertex by other edges. By doing that, trivial dead ends (dead ends of length one) can be avoided. In general, this mechanism can be extended to a larger lookahead, but this implies a penalty for the strict left-to-right incrementality.

5.4 Summary

In this chapter, we described the experiments used to validate the concept of layered charts and the system MILC. There were three main goals in these experiments.

First, we wanted to investigate whether hypergraphs are a reasonable basis for the processing of spoken language. The results of the analysis of one dialog with 41 turns resulted in a reduction of initial edges of 91%, and a reduction in overall

parsing time of 93%. Thus, hypergraphs are a powerful tool, the application of which makes the analysis of large incremental word graphs reasonable.

Second, we evaluated the behavior of MILC when applied to the translation of authentic dialogs from an appointment scheduling domain. We used five dialogs in our experiments. Two of them were used to create the grammars used in the system, while the other three dialogs were only checked for dictionary coverage. Overall, the translation of spontaneously spoken utterances can be performed in sixfold real-time. The English output is to 60.4% approximately correct, which means that the central pragmatic and propositional content was translated correctly. However, the style of the translations can only be described as very rough.

Finally, we investigated if and to what degree incremental approaches to spoken natural language processing are inferior to non-incremental approaches, as they have to cope with an input which is at least ten times larger. At least for the present case, we showed that an incremental version is not significantly slower than a non-incremental version. The difference in processing time was approximately 17%. Even if we have to concede that we used a oversimplifying model for the non-incremental version, which neglected all kinds of optimizations, we can nevertheless conclude that incremental approaches should be taken into account more in the future, since they offer tremendous methodological advantages while incurring minor penalties in performance.

Chapter 6

Conclusion and Outlook

Human language understanding works incrementally.

This insight formed the starting point for the work presented in this monograph. We have tried to demonstrate that using analogous principles in natural language processing is both desirable and reasonable. It is desirable because only the integrated processing of small parts of the input allows the construction of highly sophisticated systems for man-machine communication or support systems for human communication. Again, we can use dialog systems as an example that would be able to interrupt the user in important cases — just like humans do — or applications aiming at a simultaneous translation of spoken language.

Both the power of computers in general and the power of speech recognition systems have recently reached a level that enables an incremental processing throughout. Today, it is possible to connect a set of workstations in order to reduce the overhead introduced by incremental algorithms to a tractable amount. Our experiments show that incremental methods are roughly an order of magnitude more difficult than corresponding non-incremental approaches. Nevertheless, we estimate the processing time to be only three times higher than highly optimized conventional programs.

On the other hand, the accuracy of speech recognition systems has increased during the past few years to a degree that makes processing of spontaneous speech feasible. The dialogs recorded within the Verbmobil project are only a milestone, however, since they are restricted to a narrow domain, the scheduling of business appointments. The word accuracy we achieve in the input word graphs is approximately 70%, which is much less than what the best recognizers currently deliver in an evaluation setting for the Verbmobil domain. This value roughly corresponds to results one could expect for much less restricted speech, for instance during telephone conversations. Such a high word error rate is relativized through the use of large word graphs that encode huge amounts of alternatives in a very compact manner. Among these alternatives, the linguistic processing conducts a search for plausible interpretations.

The incremental nature of human language understanding can be shown on almost all levels of processing. We have tried to demonstrate this at least for the analysis of speech in the areas of word recognition and semantic preference modeling by providing results from psycholinguistic research. Conference interpreters "profit" from the incremental nature of the human language processor, because this

J.W. Amtrup: Incremental Speech Translation, LNAI 1735, pp. 169-174, 1999.

property allows them to simultaneously listen to source language utterances and produce target language translations at an early stage. Obviously, incrementality is in at least responsible in three ways for the human performance: First, it enables us to process almost identical parts of the input signal on several different levels of representation. Second, results from higher levels of linguistic processing can be used to block improbable alternatives on lower levels of understanding. Third, we employ feedback to generate predictions that influence some parts of the language processing facility in an expectation driven way.

Consequently, the investigation of a completely incremental speech analysis in the framework of translating spontaneously spoken language makes up the main motivation for the work described in this monograph.

However, in the process of creating a system, we also posed the goal of not having a static system that would be difficult to modify in the future. In order to analyze incremental algorithms in a qualitative way and conduct some quantitative experiments, a complete system was needed. A second aim, however, was to design and implement an open, dynamic system that could be the foundation for further architectonic studies in the framework of incremental speech processing. In particular, there is much too little known about the area of feedback between different modules, thus no commonly accepted architecture has emerged in this field. We have tried to pose as few restrictions for the implementation of alternative modules as possible.

In our view, the main contributions of this work are:

- The characterization of important properties of word graphs and the development of algorithms that facilitate the processing of even large word graphs within automatic natural language processing. Besides our critique of the common view of speech recognition evaluation and the proposal of new measures, which are not centered around recognition itself but allow a broader view of the efficiency of a whole system, we also developed algorithms to reduce the size and complexity of word graphs. In order to allow the analysis of incremental word graphs, the hypergraph mechanism provides an easy and highly efficient method, rendering the processing of input possible that, so far, has been out of reach of conventional methods.
- The design and development of an architecture schema that is broad and open enough to facilitate further research into incremental, interactive speech processing in the future. The schema consists mainly of three components that cover the important areas of data structures, representation and system integration. On the level of data structures, we introduced the layered chart, a wide-ranging extension of charts as they have been used earlier. Layered charts are capable of the distributed storage of several aspects of an input utterance with little need for synchronization. They are independent from a central storage of data or a centralized control and guarantee easy interaction within a system due to the integrated data modeling. All interaction between components is done by exchanging hyperedges, thereby preventing any kind of interface problems. Moreover, the union of all edges gives a nearly accurate picture of the current state of processing at

any given time. Layered charts extend the graph-like structure used so far mainly for speech recognition results to all levels of a system, thus they establish an integrated interpretation graph for an application.
On the level of representations, the concept and implementation of a position-invariant typed feature structure formalism with appropriateness delivers a highly efficient, declarative language for the description of linguistic objects. This language — like some of its predecessors — allows for the graph-like specification of typed feature structures. The starting point for the implementation described in this work was the assumption that a heavily pointer-oriented internal representation for feature structures is unsuitable for highly distributed, parallel systems. This is valid for inter-modular parallelism, as well as for intra-modular parallelism. Instead, we pursued an implementation using a block-oriented allocation schema for feature structures. This storage schema is extremely compact and allows for the transport of feature structures into different address spaces without a complex and time-consuming linearization and reconstruction. The formalism can be used by all components and realizes a uniform representation of diverse kinds of linguistic knowledge.
Finally, on the level of system integration, an architecture schema was developed for the construction of heterogeneous systems of components allowing a uniform view on communication. The channel-oriented design of the communication subsystem ICE guarantees a highly efficient and conceptually clear exchange of information. The class of applications targeted here consists of systems that have mainly independent, heterogeneous modules that can cooperate without the need of a central instance. A flexible configuration allows for a widespread control over the topology of a system, without interfering with the type of processing performed by individual modules. The interchange of typed messages frees the user from paying attention to infrastructural issues. The processing of an individual component is performed data-driven; distributed termination supervision is automatically done by the architecture framework.

- The complete implementation of an interpreting system for the translation of spontaneously spoken German utterances into English surface representations in the domain of appointment scheduling. This system (MILC, *M*achine *I*nterpreting with *L*ayered *C*harts) is rooted within the architecture framework just mentioned. The implementation proves that it is feasible to translate incrementally by also sticking to the chart paradigm. All types of hypotheses are viewed as potential interpretations of an interval of the input utterance. This extends not only to analysis and generation, but also to transfer which, up to now, has not been performed with the aid of chart-based methods. The incrementality is kept active even for the construction of English surface hypotheses, which is evident from the stream of growing utterance hypotheses issued by MILC.
  As important as the successful implementation of a complete working system is the possibility for changes to the system. We show an example of this strategy by integrating the recognition and translation of idioms into the system. The idiom processing is performed orthogonally to the main information flow. Its main pur-

pose is to show what the shape of future components could be. Idioms that have been recognized are treated preferentially in the rest of the system and are used to restrict search spaces that otherwise would have to be examined fruitlessly.

Within this experimental framework, several extensions offer themselves that affect the efficiency of the system and the bandwidth of phenomena covered. Immediately possible would be the exploitation of further information prevalent in the speech signal that is not being used directly by the word recognition process. In particular, the integration of prosodic evidence in order to find phrase boundaries and to assign stress patterns will have a relevant influence to the quality of translations and the speed with which they are obtained. Mode information may be used for disambiguation. Additionally, the establishment of a dialog component seems to be the next logical step, because it would enable the system to handle complete dialogs. On one hand, this would lead to better analysis results by using an additional knowledge source for disambiguation, and on the other hand, it would raise the quality of translation by employing dialogic phenomena. A further extension, which is easy and powerful simultaneously, would be the incorporation of example-based translations. Those may be treated mostly like idioms are right now, and in extreme cases may lead to very fast and accurate translations at least for parts of utterances.

For principle reasons of architecture, the coupling of the word recognizer with modules for linguistic analysis would be fruitful in order to investigate the influence of linguistic processes on speech recognition. This is especially true if restrictions from semantics or dialog modeling are applied, which have found little attention so far.

Finally, a dynamic unfolding of search spaces should only be done if the information present is not sufficient to compute a successful analysis. The approach we want to explore in this direction operates incrementally on the best path through the part of a graph that has been visited so far. The scores of edges are modeled in a way that allows to recur to acoustic hypotheses that have not yet been taken into consideration.

In a broader context, the range of applications, which rest on the concept of a layered chart, is very big. The automatic translation of spoken language represents only a small portion of the possibilities, albeit, a complex one. Besides that, access methods to information systems (e.g. data bases) or applications in natural language control (e.g. in robotics) are feasible as well.

But we don't need to restrict the applications to the processing of spoken language. Written text can be processed as well. The graph-like representation allows the modeling of ambiguity without further work; especially in the area of highly modular systems designed to handle multi-lingual input, a common architecture schema is a major advantage, if an application has to be adapted to several languages in a short period of time.

An architecture like the one we presented in this work makes it possible to store different aspects of information and to use these different aspects consistently and efficiently within a system. Restricted to natural language processing, we argued

for the incorporation of dialog phenomena that can be viewed as an easy extension in order to model supra-utterance phenomena. Similarly, prosodic knowledge can be represented as hypotheses about intervals in time of an utterance.

Viewed from a broader angle, the effect of a graph-oriented method of representation is not restricted to language-inherent properties. The extension to multi-modal input and output, as well as to the representation of events, can be achieved. This is immediately obvious for all kinds of input information that is thematically and chronologically related to the speech input in a narrower sense. Deictic gestures of an user, for instance, can be easily integrated into the course of a system, be they mouse movements, eye or hand gestures. They could be recognized — similarly to idioms — in a specialized component that delivers a description of the action in the form of a feature structure and its extension in time. Obviously, the other components of the system have to be modified accordingly, in order to be able to process that additional information.

A further type of additional input channel is given by the lip movements of a speaker. The chronological and conceptual relation between the original input (a word graph) and the input in another modality is immediately clear. The information about the lip configuration can be used for disambiguation. An articulatory model could be constructed form a word graph, which then could be matched with the actual lip movements issued. Again, the additional information represents hypotheses about intervals in time during the input speech.

Much more complex, but nevertheless possible, is the integration of arbitrary objects and their states that change over time. The starting point for such an endeavor would be the extension of the coverage of a layered chart over the boundaries of one utterance and directed toward a continuous time model. In order to restrict the amount of data necessary, it is important to develop a focus model that describes which area in time is currently most interesting. Once the focus changes, irrelevant information can be removed from the chart and only central hypotheses prevail.

The second problem here is the introduction of a method for combining linguistic and extra-linguistic knowledge. We can't do this here for reasons of time and space. However, a graph-like representation is very suitable to model relations in time by mapping them to topological properties. In the simplest case, relations can be viewed as an intersection of vertex sets or as a computation of the reachability relation.

Moreover, the representation of non-linguistic knowledge has to be tightly integrated into the preexisting type lattice. The goal would be to prevent an uncontrollable interaction between purely linguistic facts and extra-linguistic information. Objects and state descriptions must not interfere with linguistic operations. Instead, the processing of such heterogeneous knowledge has to be made explicit. This does not exclude, however, the application of methods developed in this monograph to the processing of edges of different origin. The multiple layers of a layered chart are suitable for encapsulating different aspects of information without preventing their integrated processing. Just to mention one example, object and procedural references can be made much easier, for instance in a maintenance scenario in which

an unexperienced user attempts to refill motor oil. Deictic expressions ("Not that one!" if the user tries to open the cooling system cap instead of the oil cap) can be generated and analyzed more easily. In this hypothetical example, the dialog focused around the oil cap, but the object modeling also knows about other caps (like the cooling cap). The chronological relation between the request to open the oil cap and the action of the user (trying to open the cooling cap) enables the system to establish a restriction of the reference and the system is able to correct the user's action. In such a scenario, the relevance of incremental operation surfaces again. The system should be able to warn the user at once in order to prevent mistakes during the maintenance operations.

All hypothetical areas of applications presented here exhibit certain common properties: The primary dimension of the input is time. Complex objects (linguistic descriptions, representations of actions) are used to represent knowledge. The employment of incremental techniques is necessary or at least desirable. A data structure like a layered chart, which is oriented toward graph like structures, provides an excellent starting point for the investigation of such phenomena, given the architectonic environment we described.

Bibliography

Abney, Steven. 1991. Parsing By Chunks. In Robert Berwick, Steven Abney and Carol Tenny, editors, *Principle-Based Parsing.* Kluwer Academic Publishers, Dordrecht.

Abney, Steven. 1996. Partial Parsing via Finite-State Cascades. *In Proceedings of the ESSLLI '96 Robust Parsing Workshop.*

Agnas, Marie-Susanne, Hiyan Alshawi, Ivan Bretan and David Carter *et al.* 1994. Spoken Language Translator: First-Year Report. Research Report R94:03, SICS, Stockholm, January.

Aigner, Martin. 1984. *Graphentheorie: Eine Entwicklung aus dem 4-Farben Problem.* Teubner Studienbu¨cher: Mathematik. Stuttgart: Teubner.

Ait-Kaci, Hassan, Robert Boyer, Patrick Lincoln and Roger Nasr. 1989. Efficient Implementation of Lattice Operations. *ACM Transactions on Programming Languages and Systems*, 11(1): 115-146, January.

Alexandersson, Jan, Elisabeth Maier and Norbert Reithinger. 1995. A Robust and Efficient Three-Layered Dialog Component for a Speech-to-Speech Translation System. In *Proc. of the 7 th EACL*, pages 188-193, Bergen, Norway. Also available as: Verbmobil-ReportNo. 50, DFKI GmbH, December 1994.

Alexandersson, Jan, Norbert Reithinger and Elisabeth Maier. 1997. Insights into the Dialogue Processing of Verbmobil. In *Proc. of the 5th Conference on Applied Natural Language Processing*, Washington, D.C.

Allen, James F. 1987. *Natural Language Understanding.* The Ben-jamin/Cummings Series in Computer Science. Menlo Park, CA: Ben-jamin/Cummings.

Allen, James F, Bradford W. Miller, Eric K. Ringger and Teresa Sikorski. 1996. A Robust System for Natural Spoken Dialogue. In *Proc. of the 34 nd ACL*, pages 62-70, Santa Cruz, CA, June.

Allen, James F, LenhartK. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio and David R. Traum. 1994. The TRAINS Project: A case study in building a conversational planning agent. TRAINS Technical Note 94-3, University of Rochester, Rochester, NY, September.

Amtrup, Jan W. 1992. Parallele Strukturanalyse Naturlicher Sprache mit Transput-ern. ASL-TR 44-92/UHH, Univ. of Hamburg.

J.W. Amtrup: Incremental Speech Translation, LNAI 1735, pp. 175-192, 1999.

Amtrup, Jan W. 1994a. ICE-Intarc Communication Environment: Design und Spezifikation. Verbmobil Memo 48, Univ. of Hamburg, September.

Amtrup, Jan W. 1994b. Transfer and Architecture: Views from Chart Parsing. Verbmobil Report 8, Univ. of Hamburg, March.

Amtrup, Jan W. 1995a. Chart-based Incremental Transfer in Machine Translation. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '95,* pages 188-195, Leu-ven, Belgium, July.

Amtrup, Jan W. 1995b. ICE-Intarc Communication Environment: User's Guide and Reference Manual. Version 1.4. Verbmobil Technical Document 14, Univ. of Hamburg, December.

Amtrup, Jan W. 1995c. Parallel Parsing: Different Distribution Schemata for Charts. In *Proceedings of the 4th International Workshop on Parsing Technologies (IWPT95),* pages 12-13, Prague, September. Charles University.

Amtrup, Jan W. 1997a. ICE: A Communication Environment for Natural Language Processing. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA97),* Las Vegas, NV, July.

Amtrup, Jan W. 1997b. Layered Charts for Speech Translation. *In Proceedings of the Seventh International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '97,* Santa Fe, NM, July.

Amtrup, Jan W. 1997c. Perspectives for Incremental MT with Charts. In Christa Hauenschild and Susanne Heizmann, editors, *Machine Translation and Translation Theory. Perspectives of Co-operation,* Text, Translation, Computational Processing (TTCP), number 1. Moutonde Gruyter.

Amtrup, Jan W. 1998. Incremental Speech Translation: A Layered Chart Approach. In *28. Jahrestagung der Gesellschaftfur Informatik,* Magdeburg, September.

Amtrup, Jan W. and Jorg Benra. 1996. Communication in Large Distributed AI Systems for Natural Language Processing. *InProc. of the 16 th COLING,* pages 35-40, Copenhagen, Denmark, August. Center for Sprogteknologi.

Amtrup, Jan W., Henrik Heine and Uwe Jost. 1996. What's in a Word Graph — Evaluation and Enhancement of Word Lattices. Verbmobil Report 186, Univ. of Hamburg, Hamburg, December.

Amtrup, Jan W., Henrik Heine and Uwe Jost. 1997. What's in a Word Graph — Evaluation and Enhancement of Word Lattices. In *Proc. of Eurospeech 1997,* Rhodes, Greece, September.

Amtrup, Jan W and Susanne J. Jekat. 1995. Segmentation of Spoken Language for NLP. In *KI95-Activities: Workshops, Posters, Demos,* pages 298-299, Bielefeld, September.

Amtrup, Jan W and Volker Weber. 1998. Time Mapping with Hypergraphs. In *Proc. of the 17 th COLING,* Montreal, Canada.

Anderson, Linda. 1994. Simultaneous Interpretation: Contextual and Translation Aspects. In Sylvie Lambert and Barbara Moser-Mercer, editors, *Bridging the*

Gap: Empirical Research in Simultaneous Interpretation. John Benjamins Publishing Co, pages 101-120.

Aubert, Xavier and Hermann Ney. 1995. Large Vocabulary Continuous Speech Recognition Using Word Graphs. *In ICASSP 95.*

Ausiello, Giorgio, Giuseppe F. Italiano, Alberto Marchetti Marchetti-Spaccamela and Umberto Nanni. 1991. Incremental Algorithms for Minimal Length Paths. *Journal of Algorithms,* 12:615-638.

Backofen, Rolf, Lutz Euler and Gu¨nther Gorz. 1991. Distributed Disjunctions in LIFER. In *Proc. International Workshop on Processing Declarative Knowledge,* pages 161-170, Berlin, Heidelberg, New York. Springer Verlag.

Batliner, Anton, Susanne Burger and Andreas Kiessling. 1994. Ausergrammatische Phanomene in der Spontansprache. VM Technisches Dokument 2, Univ. Mu¨nchen.

Beaven, J.L. 1992. *Lexicalist Unification BasedMachine Translation.* Ph.D. thesis, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK.

Beskow, Bjorn. 1993. Unification Based Transfer: Multilingual Support for Translation and Writing. Draft, Uppsala University, Uppsala, Sweden, February.

Billot, Sylvie and Bernard Lang. 1989. The Structure of Shared Forests in Ambiguous Parsing. In *Proc. of the 27 th ACL,* pages 143-151, Vancouver, June.

Block, H. U. 1997. The Language Components in Verbmobil. *In Proc. ICASSP '97,* pages 79-82, Munich, Germany, April.

Boddy, Mark and Thomas L. Dean. 1994. Deliberation Scheduling for Problem Solving in Time-Constrained Environments. *Artificial Intelligence,* 67(2):245-285.

Boitet, Christian and Mark Seligman 1994. The "Whiteboard" Architecture: A Way to Integrate Heterogeneous Components of NLP systems. In *COLING-94: The 15th International Conference on Computational Linguistics,* Kyoto, Japan.

Brandstadt, Andreas, 1994. "Ku¨rzeste Wege". *In GraphenundAlgorithmen,* chapter 5, pages 106-123. Stuttgart: Teubner.

Bresnan, Joan, editor. 1982. *The Mental Representation of Grammatical Relation.* Cambridge, MA: MIT Press.

Brew, Chris. 1992. Letting the Cat out of the Bag: Generation for Shake-and-Bake MT. In *COLING-92: The 15th International Conference on Computational Linguistics,* pages 610-616, Nantes, France.

Brietzmann, Astrid. 1992. "Reif fur die Insel". Syntaktische Analyse naturlich gesprochener Sprache durch bidirektionals Chart-Parsing. In Helmut Mangold, editor, *Sprachliche Mensch-Maschine-Kommunikation.* R. Oldenbourg Verlag, pages 103-116.

Briscoe, Edward J. 1987. *Modelling Human Speech Comprehension: A Computational Approach.* Wiley.

Brown, P., J. Cocke, S. A. Della Pietra, Felinek, J. D. F. Lafferty, R. L. Mercer and P. S. Roossin. 1990. A Statistical Approach to Machine Translation. *Computational Linguistics,* 16:79-85.

Brown, Ralf and Robert Frederking. 1995. Applying Statistical English Language Modelling to Symbolic Machine Translation. In *TMI95P*, TMI95L.

Bub, Thomas, Wolfgang Wahlster and Alex Waibel. 1997. Verbmobil: The Combination of Deep and Shallow Processing for Spontaneous Speech Translation. In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pages 1/71-1/74, Munich, Germany.

Burns, Alan. 1988. *Programming in OCCAM2.* Reading, Ma.: Addison-Wesley.

Buschbeck-Wolf, Bianka et al. 1995. Transfer in the Verbmobil Demonstrator. Technical report, IAI, Saarbru¨cken.

Carlson, Lauri and Maria Vilkuna. 1990. Independent Transfer Using Graph Unification. In *Proc. of the 13 th COLING*, pages 3/60-3/63, Helsinki, Finland.

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures.* Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press.

Carpenter, Bob and Gerald Penn. 1998. ALE - The Attribute Logic Engine User's Guide. Version 3.0 Beta. Technical report, Bell Labs/Univ. Tubingen, March.

Carpenter, Bob and Yan Qu. 1995. An Abstract Machine for Attribute-Value Logics. In *Proceedings of the 4th International Workshop on Parsing Technologies (IWPT95)*, pages 59-70, Prague. Charles University.

Carter et al., David. 1997. Translation Methodology in the Spoken Language Translator: An Evaluation. In *ACL Workshop on Spoken Language Translation.*

Carver, N. and V Lesser. 1994. The Evolution of Blackboard Control Architectures. *Expert Systems with Applications*, 7(1): 1-30.

Caspari, Rudolf and Ludwig Schmid. 1994. Parsing und Generierung in TrUG. Verbmobil-Report 40, Siemens AG.

Chen, Wai-Kai. 1971. *Applied Graph Theory.* Applied Mathematics and Mechanics, volume 13, edited by. Amsterdam: North Holland.

Chernov, G. V 1994. Message Redundancy and Message Anticipation in Simultaneous Interpretation. In Lambert and Moser-Mercer, editors, *Bridging the Gap: Empirical Research in Simultaneous Interpretation.* John Benjamins, pages 139-153.

Cheston, Grant A. 1976. *Incremental Algorithms in Graph Theory.* Ph.D. thesis, Univ. of Toronto, March.

Chomsky, Noam. 1959. On certain formal properties of grammars. *Information and Control*, 2(2):137-167.

Chomsky, Noam. 1995. *The Minimalist Program.* Current Studies in Linguistics, number 28. Cambridge, MA: The MIT Press.

Cohen, P. R., A. Cheyer, M. Wang and S. C. Baeg. 1994. An Open Agent Architecture. In *Proc. of AAAI-94*, pages 1-8, Stanford, CA.

Copestake, Ann, Dan Flickinger, Rob Malouf, Susanne Riehemann and Ivan Sag. 1995. Translation using Minimal Recursion Semantics. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '95,* Leuven, Belgium.

Corbin, John R. 1990. *The Art of Distributed Applications.* Sun Technical Reference Library. New York: Springer-Verlag.

Cormen, Thomas H, Charles E. Leiserson and Roanld L. Rivest. 1990. *Introduction to Algorithms.* Cambridge, MA: MIT Press.

Denecke, Matthias. 1997. A Programmable Multi-Blackboard Architecture for Dialogue Processing Systems. In *ACL97P,* ACL97L.

Diagne, Abdel Kader, Walter Kasper and Hans-Ulrich Krieger. 1995. Distributed Parsing with HPSG Grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies (IWPT95),* pages 79-86, Prague, September. Charles University.

Dorna, Michael. 1992. Erweiterung der Constraint-Logiksprache CUF um ein Typensystem. Master's thesis, Univ. of Stuttgart.

Dorna, Michael and Martin C. Emele. 1996. Efficient Implementation of a Semantic-based Transfer Approach. In *Proc. of the 12th ECAI,* Budapest, Hungary, August.

Dorr, Bonnie Jean. 1993. *Machine Translation: A View from the Lexicon.* Cambridge, MA.: MIT Press.

Doyle, Jon. 1979. A Truth Maintenance System. *Artificial Intelligence,* 12:231–272.

Earley, Jay. 1970. An Efficient Context-Free Parsing Algorithm. *Communications oftheACM,* 13:94-102.

Eberle, Kurt, Walter Kasper and Christian Rohrer. 1992. Contextual Constraints for MT. In *Proc. of the 4 th Int. Conf. on Theoretical and Methodological Issues in Machine Translation,* pages 213-224, Montreal, Canada, June.

Eisele, Andreas and Jochen Dorre. 1988. Unification of Disjunctive Feature Structures. In *Proc. of the 26 th ACL,* Buffalo, NY, June.

Elsner, Anja and Alexandra Klein. 1996. Erkennung des prosodischen Fokus und die Anwendung im dialogaktbasierten Transfer. Verbmobil Memo 107, Univer-sitat Bonn, Universitat Hamburg.

Emele, Martin, Ulrich Heid, Stefan Momma and Remi Zajac. 1991. Interactions between Linguistic Constraints: Procedural vs. Declarative Approaches. *Machine Translation,* pages 61-98.

Emele, Martin E. and Remi Zajac. 1990. Typed Unification Grammars. In *Proc. of the 13 th COLING,* pages 293-298, Helsinki, Finland.

Engelmore, Robert and Tony Morgan. 1988. *Blackboard Systems.* Reading, MA: Addison-Wesley Publishing Company.

Erbach, G. and B. Krenn. 1994. Idioms and Support-Verb ConstructionsIn J. Ner-bonne, K. Netter and C. Pollard, editors, *German Grammar in HPSG.* CSLI, Stanford, CA.

Erman, Lee D., Frederick Hayes-Roth, Victor R. Lesser and Raj Reddy. 1980. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys,* 12(2):213-253.

Fink, Gernot A., Franz Kummert and Gerhard Sagerer. 1994. A Close High-Level Interaction Scheme for Recognition and Interpretation of Speech. In *Proc. ICSLP-94,* pages 2183-2186, Yokohama, Japan.

Gondran, Michel and Michel Minoux. 1984. *Graphs and algorithms.* Wiley-Interscience Series in Discrete Mathematics. Chichester: John Wiley & Sons.

Gorz, Gu¨nther. 1988. *Strukturanalyse natu¨rlicher Sprache.* Bonn: Addison Wesley.

Gorz, Gu¨nther. 1993. Kognitiv orientierte Architekturen fur die Sprachverar-beitung. Technical Report ASL-TR-39-92, Universitat Erlangen-Nurnberg, February.

Gorz, Gu¨nther and Marcus Kesseler. 1994. Anytime Algorithms for Speech Parsing? In *COLING-94: The 15th International Conference on Computational Linguistics,* Kyoto, Japan.

Gorz, Gu¨nther, Marcus Kesseler, Jorg Spilker and Hans Weber. 1996. Research on Architectures for Integrated Speech/Language Systems in Verbmobil. In *Proc. of the 16 th COLING,* pages 484-489, Copenhagen, Denmark, August.

Grabski, Michael. 1990. Transfer Statements as Conditional Constraints. WP 18/90. Eurotra-d working papers, IAI, Saarbru¨cken.

Graham, Ian and Tim King. 1990. *The Transputer Handbook.* New York, London etal.: Prentice Hall.

Guard, J.R. 1964. Automated Logic for Semi-Automated Mathematics. Scientific Report 1, Air Force Cambridge Research Laboratory, Bedford, MD.

Hager, Jochen and Martin Moser. 1989. An Approach to Parallel Unification Using Transputers. In *German Workshop on Artificial Intelligence,* pages 83-91, Eringerfeld.

Hahn, Walther v. 1992. Von der Verknu¨pfung zur Integration: Kontrollstrate-gie oder kognitive Architektur. In *Procceedings ofKONVENS92,* pages 1-10, Berlin. Springer Verlag.

Hahn, Walther v. and Jan W Amtrup. 1996. Speech-to-Speech Translation: The Project Verbmobil. *InProceedings ofSPECOM96,* pages 51-56, St. Petersburg, October.

Hanrieder, Gerhard. 1996. *Inkrementelles Parsing gesprochener Sprache mit einer linksassoziativen Unifikationsgrammatik.* DisKi, number 140. St. Augustin: Infix.

Harary, Frank. 1974. *Graphentheorie.* Mu¨nchen, Wien: R. Oldenbourg Verlag.

Harbusch, Karin. 1990. Constraining Tree Adjoining Grammars by Unification. In *Proc. of the 13th COLING,* pages 167-172, Helsinki, Finland.

Haruno,Masahiko, YasuharuDen, YujiMastumotoandMakatoNagao. 1993. Bidirectional Chart Generation of Natural Language Texts. In *Proc. of AAAI-93,* pages 350-356.

Hauenschild, Christa. 1985. KIT/NASEV oder die Problematik des Transfers bei der Maschinellen Ubersetzung. KIT Report 29, Technische Universitat Berlin, Berlin, November.

Hauenschild, Christa and Birte Prahl. 1994. Konzept Translationsprobleme -

Trans-lationsstrategien. Technical report, Univ. of Hildesheim.

Hauenstein, Andreas. 1996. *Aussprachewo¨rterbu¨cher zur automatischen Spracherkennung*. DISKI Dissertationen zur Ku¨nstlichen Intelligenz, number 133. St. Augustin: infix.

Hauenstein, Andreas and Hans Weber. 1994. An Investigation of Tightly Coupled Speech Language Interfaces Using an Unification Grammar. In *Proceedings of the Workshop on Integration of Natural Language and Speech Processing at AAAI '94*, pages 42–50, Seattle, WA.

Hayes-Roth, Barbara. 1995. A Blackboard Architecture for Control. *Artificial Intelligence*, 26:251–321.

Hillis, W.D. 1985. *The Connection Machine*. Cambridge, MA: MIT Press.

Hoare, Charles A. Richard. 1978. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August.

Hopcroft, J. and J. Ullman. 1979. *introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Horacek, Helmut. 1993. Sprachgenerierung: Planungsvrefahren und Architektur-modelle. *K u¨nstliche Intelligenz*, 7(2):8–13.

Huebener, Kai, Uwe Jost and Henrik Heine. 1996. Speech Recognition for Spontaneously Spoken German Dialogs. In *ICSLP96*, Philadelphia, PA.

Hutchins, John. 1994. Research methods and system designs in machine translation. In *Machine Translation: Ten Years On*, pages 5–1 – 5–16, Cranfield, UK, November.

Hutchins, W. John. 1986. *Machine Translation. Past, Present and Future*. New York: Horwood.

Hutchins, W. John and Harold L. Somers. 1992. *An Introduction to Machine Translation*. London: Academic Press.

Jagannathan, V., R. Dodhiawala and L. Baum (eds.). 1989. *Blackboard Architectures and Applications*. Boston, MA: Academic Press.

Jekat, Susanne, Alexandra Klein, Elisabeth Maier, Ilona Maleck, Marion Mast and Joachim Quantz. 1995. Dialogue Acts in Verbmobil. Verbmobil Report 65, Universita¨t Hamburg, DFKI GmbH, Universita¨t Erlangen, TU Berlin.

Jekat, Susanne J. 1997. Automatic Interpretation of Dialogue Acts. In Christa Hauenschild and Susanne Heizmann, editors, *Machine Translation and Translation Theory. Perspectives of Co-operation*, Text, Translation, Computational Processing (TTCP), number 1. Mouton de Gruyter.

Jekat, Susanne J., Alexandra Klein, Elisabeth Maier, Ilona Maleck, Marion Mast and J. Joachim Quantz. 1995. Dialogakte in Verbmobil. Verbmobil Technisches Dokument 26, Universita¨t Hamburg.

Joshi, Aravind K. 1985. How much Context-Sensitivity is Necessary for Characterizing Structural Descriptions—Tree Adjoining GrammarsIn D. Dowty, L. Kart-tunen and A. Zwicky, editors, *Natural Language Processing — Theoretical, Computational and Psychological Perspectives*. Cambridge University Press, New York.

Jost, Uwe. 1997. System- und Modulevaluation. Verbmobil-Memo 125, Univer-sita¨t Hamburg.

Juola, Patrick. 1994. Self-Organizing Machine Translation: Example-Driven Induction of Transfer Functions. Technical Report CU-CS722-94, Univ. of Colorado, Boulder, CO, May.

Kaplan, R., K. Netter, J. Wedekind and A. Zaenen. 1989. Translation by Structural Correspondence. In *Proc. of the 4th EACL,* Manchester, UK.

Kaplan, Ronald M. 1973. A General Syntactic Processor. In *Natural Language Processing.* Algorithmic Press, Inc., New York, pages 193-241.

Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, MA, pages 173-281.

Kaplan, Ronald M. and John T. Maxwell. 1989. An Overview of Disjunctive Constraint Satisfaction. In *Proc. International Parsing Workshop,* pages 18-27, Pittsburgh, PA. Carnegie Mellon University.

Karlgren, Jussi. 1994. Mumbling — User-Driven Cooperative Interaction. Technical report, SICS, Stockholm, January.

Kasper, W., H.-U. Krieger, J. Spilker and H. Weber. 1996. From Word Hypotheses to Logical Form: An Efficient Interleaved Approach. In *Proc. ofKONVENS96.*

Kasper, Walter and Hans-Ulrich Krieger. 1996. Integration of Prosodic and Grammatical Information in the Analysis of Dialogs. In *KI-96: Advances in Artificial Intelligence. 20th Annual German Conference on Artificial Intelligence,* pages 163-174, Berlin, September. Springer Verlag.

Katoh, Naoto and Teruaki Aizawa. 1994. Machine Translation of Sentences with Fixed Expressions. In *Proc. of the 4 th Conference on Applied Natural Language Processing,* pages 28-33, Stuttgart, Germany, October.

Kay, M., J.M. Gawron and P. Norvig. 1991. *Verbmobil: A Translation System for Face-to-Face Dialog.* CSLI.

Kay, Martin. 1973. The MIND System. InR. Rustin, editor, *Natural Language Processing.* Algorithmic Press, New York, pages 155-188.

Kay, Martin. 1979. Functional Grammar. In C. Chiarelloet*et al,* editor, *Proc. 5th Annual Meeting of the Berekeley Linguistic Society,* pages 142-158, Berkeley, CA.

Kay, Martin. 1980. Algorithmic Schemata and Data Structures in Syntactic Processing. Technical Report CSL-80-12, Xerox Palo Alto Research Center, Palo Alto, CA.

Kay, Martin. 1984. Functional Unification Grammar: A Formalism for Machine Translation. In *Proc. of the 10 th COLING,* pages 75-78, Stanford, CA.

Kay, Martin. 1996. Chart Generation. In *Proc. of the 34 nd ACL,* pages 200-204, Santa Cruz, CA, June.

Kesseler, Marcus. 1994. Distributed Control in Verbmobil. Verbmobil Report 24, Univ. of Erlangen-Nurnberg, August.

Kesseler, Marcus P. 1990. TransScheme: Entwurf und Implementierung eines verteilten Scheme-Lisp Systems fur Transputernetzwerke. Master's thesis, Uni-versitat Erlangen-Nurnberg, October.

Kiefer, Bernd and Thomas Fettig. 1993. FEGRAMED: An Interactive Graphics Editor for Feature Structures. DFKI-Report, April.

Kikui, Gen-ichiro. 1992. Feature Structure Based Semantic Head Driven Generation. In *COLING-92: The 15th International Conference on Computational Linguistics*, pages 32-38, Nantes, France.

Kilbury, James. 1985. Chart Parsing and the Earley Algorithm. KIT-Report 24, Projektgruppe Ku¨nstliche Intellligenz und Textverstehen.

Kilger, Anne. 1994. Using UTAGS for Incremental and Parallel Generation. *Computational Intelligence*, 10(4):591-603, November.

Kinoshita, Satoshi, John Phillips and Jun-ichi Tsujii. 1992. Interaction between Structural Changes in Machine Translation. In *COLING-92: The 15th International Conference on Computational Linguistics*, pages 679-685, Nantes, France.

Kitano, H. 1990. #DMDIALOG: A Speech-to-Speech Dialouge Translation System. *Machine Translation*, 5.

Kitano, Hiroaki. 1994. *Speech-to-Speech Translation: A Massively Parallel Memory-Based Approach.* Boston: Kluwer Academic Publishers.

Klein, Alexandra, Susanne J. Jekat and Jan W. Amtrup. 1996. Inkrementelle und erwartungsgesteuerte Verarbeitung beim Maschinellen Dolmetschen. In *Proceedings der zweiten Fachtagung der Gesellschaft fur Kognitionswissenschaft*, pages 68-70, Hamburg, March.

Knight, K. 1989. Unification: A Multi-Disciplinary Survey. *ACM Computer Surveys, 21:98-124.*

Knight, Kevin, I. Chander, M. Haines, V Hatzivassiloglou, E. H. Hovy, M. Iida, S. K. Luk, A. Okumura, R. A. Whitney and K. Yamada. 1994. Integrating Knowledge Sources and Statistics inMT. In *Proceedings of the 1stAMTA Conference*, Columbia, MD.

Konieczny, Lars. 1996. *Human Sentence Processing: A Semantics-Oriented Parsing Approach.* Ph.D. thesis, Albert-Ludwigs-Universitat, Freiburg.

Konig, Esther. 1994. Syntactic-Head-Driven Generation. In *COLING-94: The 15th International Conference on Computational Linguistics*, Kyoto, Japan.

Krieger, Hans-Ulrich. 1995. *TDC—A Type Description Language for Constraint-Based Grammars. Foundations, Implementation, and Applications.* Ph.D. thesis, Universitat des Saarlandes, Department of Computer Science, September.

Ku¨nzli, Alexander and Barbara Moser-Mercer. 1995. Human Strategies for Translation and Interpretation. In*KI95-Activities: Workshops, Posters, Demos*, pages 304-306, Bielefeld.

Lavie, Alon, Alex Waibel, Lori Levin, Michael Finke, Donna Gates, Marsal Gavalda, Torsten Zeppenfeld and Puming Zhan. 1997. JANUS III: Speech-to-Speech Translation in Multiple Languages. In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP,*

Munich, Germany.

Lehning, Michael. 1996. Evaluierung von signalnahen Spracherkennungssyste-men fuer deutsche Spontansprache. Verbmobil Report 161, TU Braunschweig, www.dfki.uni-sb.de/verbmobil.

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

Light, Marc. 1996. CHUMP: Partial Parsing and Underspecified Representations. In *Proceedings of the ECAI-96 Workshop: Corpus-Oriented Semantic Analysis.*

Luckhardt, Heinz-Dirk. 1987. *Der Transfer in der maschinellen Sprachuberset-zung*. Sprache und Information. Tubingen: Niemeyer.

Marchetti-Spaccamela, Alberto, Umberto Nanni and Hans Rohnert. 1992. On-line Graph Algorithms for Incremental Compilation. Technical Report TR-92-056, ICSI.

Marcus, M. 1980.*A Theory of Syntactic Recognition for Natural Language.* Cambridge, MA: MIT Press.

Marslen-Wilson, William D. 1987. Functional Parallelism in Spoken Word Recognition. *Cognition*, 25:71-102.

Marslen-Wilson, William D. and Lorraine K. Tyler. 1980. The Temporal Structure of Spoken Language Understanding. *Cognition*, 8:1-71.

Marslen-Wilson, William D. and A. Welsh. 1978. Processing Interactions During Word Recognition in Continuous Speech. *Cognitive Psychology*, 10:29-63.

Matsubara, Shiegki and Yasuyoshi Inagaki. 1997a. Incremental Transfer in English-Japanese Machine Translation. *IEICE Transactions on Information and Systems*, 80(11): 1122-1129, November.

Matsubara, Shiegki and Yasuyoshi Inagaki. 1997b. Utilizing Extra-Grammatical Phenomena in Incremental English-Japanese Machine Translation. In *Proceedings of the Seventh International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '97,* pages 31-38, Santa Fe, NM, July.

Mattern, Friedemann. 1987. Algorithms for Distributed Termination Detection. Technical Report 20/87, SFB 124, Kaiserslautern.

Mayer, Otto. 1986. *Syntaxanalyse.* Reihe Informatik, number 27. Mannheim: Bibliographisches Institut.

McClelland, J. L. and J. L. Elman. 1986. The TRACE model of speech perception. *Cognitive Psychology*, 18:1-86.

McHugh, James A. 1990. *Algorithmic Graph Theory*. Englewood Cliffs, NJ: Prentice Hall.

Melamed, I. Dan. 1998. Word-to-Word Models of Translational Equivalence. IRCS Technical Report 98-08, Uinv. Pennsylvania.

Menzel, Wolfgang. 1994. Parsing of Spoken Language under Time Constraints. In T. Cohn, editor, *Proc. of the 11th ECAI*, pages 560-564.

Menzel, Wolfgang. 1998. Constraint Satisfaction for Robust Parsing of Spoken Language. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(1):77-89.

Milward, David. 1995. Incremental Interpretation of Categorial Grammar. In *Proc. of the 7th EACL*, Bergen, Norway.

Mima, Hideki, Hitoshi Iida and Osamu Furuse. 1998. Simultaneous Interpretation Utilizing Example-Based Incremental Transfer. In *COLING98P*, COLING98L.

Morimoto, T., M. Suzuki, T. Takazewa, F. Yato, S. Sagayama, T. Tashiro andM. Na-gata. 1993. ATR's Speech translation System: ASURA. In *Proc. ofEurospeech 1993.*

Morimoto, Tsuyoshi, Masami Suzuki, Tosiyuki Takezawa, Genichiro Kikui, Masaaki Nagata and Mutsuko Tomokiyo. 1992. A Spoken Language Translation System: SL-TRANS2. In *COLING-92: The 15th International Conference on Computational Linguistics*, pages 1048-1052, Nantes, France.

Mouaddib, Abdel-illah and Shlomo Zilberstein. 1995. Knowledge-Based Anytime Computation. In *Proc. of the 14th IJCAI*, pages 775-781, Montreal, Canada.

Nagao, M. and J. Tsujii. 1986. The Transfer Phase of the Mu Machine Translation System. In *Proc. of the 11th COLING*, pages 97-103, Bonn, FRG.

Nederhof, Mark-Jan and Giorgio Satta. 1994. An Extended Theory of Head-Driven Parsing. *In Proc. of the 32 nd ACL*, Las Cruces, NM.

Niemann, Heinrich, Elmar Noth, Andreas Kiessling, Ralf Kompe and Anton Bat-liner. 1997. Prosodic Processing and its Use in Verbmobil. In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP.*

Niessen, S., S. Vogel, H. Ney and C Tillmann. 1998. A DP-Based Search Algorithm for Statistical Machine Translation. In *COLING98P*, COLING98L.

Nirenburg, Sergei. 1987. Knowledge and Choices in Machine Translation. In Sergei Nirenburg, editor, *Machine Translation: Theoretical and Methodological Issues.* Cambridge University Press, pages 1-21.

Nirenburg, Sergei. 1992. *Machine Translation. A Knowledge-based Approach.* San Mateo, CA: Kaufmann.

Nirenburg, Sergei (ed.). 1993. *Progress in Machine Translation.* Amsterdam: IOS Press.

Niv, Michael. 1993. *A Computational Model of Syntactic Processing: Ambiguity Resolution from Interpretation.* Ph.D. thesis, Univ. of Pennsylvania.

Noeth, Elmar, Anton Batliner, Andreas Kiessling, Ralf Kompe and Heinrich Niemann. 1997. Prosodische Information: Begriffsbestimmung und Nutzen fur das Sprachverstehen. In Paulus and Wahl, editors, *Mustererkennung 1997*, In-formatik Aktuell, Heidelberg. Springer Verlag.

Noord, Gertjan van. 1990. Reversible Unification Based Machine Translation. In *Proc. of the 13th COLING*, pages 299-304, Helsinki, Finland.

Oerder, Martin and Hermann Ney. 1993. Word Graphs: An Efficient Interface Between Continuous-Speech Recognition and Language Understanding. In *Proceedings of the 1993 IEEE International Conference on Acoustics, Speech & Signal Processing, ICASSP*, pages II/119–II/122, Minneapolis, MN.

Oi, Kozo, Eiichiro Sumita, Osamu Furuse, Hitoshi Iida and Tetsuya Higuchi. 1994. Real-Time Spoken Language Translation Using Associative Processors. In *Proc. of the 4 th Conference on Applied Natural Language Processing*, pages 101-106, Stuttgart, Germany.

Ousterhout, John K. 1994. *Tcl and the Tk Toolkit.* Addison-Wesley.

Paterson, M. S. andM. N. Wegman. 1978. Linear Unification. *Journal of Computer and System Sciences*, 16:158-167.

Pereira, Fernando C. N and Stuart M. Shieber. 1984. The Semantics of Grammar Formalisms Seen as Computer Languages. In *Proc. of the 10 th COLING*, Stanford, CA.

Peres, L. andB. Rozoy. 1991. On the Evaluation of Distributed Termination Protocols. Rapport de Recherche 656, LRI, Paris.

Pollard, Carl and Ivan A. Sag. 1987. *Information-based Syntax and Semantics. Vol 1: Fundamentals.* Stanford, CA: CSLI Lecture Notes 13.

Pollard, Carl and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar.* Chicago, London: University of Chicago Press.

Poller, Peter. 1994. Incremental parsing with LD/TLP-TAGS. *Computational Intelligence*, 10(4):549-562, November.

Poznan´ski, V., J. L. Beaven and P. Whitelock. 1995. An Efficient Generation Algorithm for Lexicalist MT *In Proc. of the 33 nd ACL*, Cambridge, MA, June.

Prahl, Birte, Susanne Petzold, Susanne Heizmann and Christa Hauenschild. 1995. Variable Analysetiefe und Bewertungskriterien in Verbmobil: Translationswis-senschaftliche Grundlagen. Verbmobil-Report 54, University of Hildesheim, Hildesheim, January.

Pyka, Claudius. 1992a. Management of Hypotheses in an Integrated Speech-Language Architecture. In *Proc. of the 10th ECAI*, pages 558-560, Vienna, Austria.

Pyka, Claudius. 1992b. Schnittstellendefinition mit ASL-DDL. ASL-TR 42-92/UHH, Univ. of Hamburg, March.

Pyka, Claudius. 1992c. Spezifikation einer Komponente. Technical Report ASL-Memo-57-92/UHH, Univ. of Hamburg, Hamburg, September.

Ramalingam, G. and T. Reps. 1992. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. Technical report, Univ. of Wisconsin - Madison.

Rayner, Manny and David Carter. 1996. Fast Parsing Using Pruning and Grammar Specialization. In *Proc. of the 34 nd ACL*, pages 223-230, Santa Cruz, CA, June.

Rayner, Manny and David Carter. 1997. Hybrid Language Processing in the Spoken Language Translator. In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, Munich, Germany. http://www.cam.sri.com/tr/crc064/paper.ps.Z.

Reinecke, Joerg. 1996. Evaluierung der signalnahen Spracherkennung. Verbmobil Memo 113, TU Braunschweig, Nov.

Reithinger, Norbert. 1992. *Eine parallele Architektur zur Inkrementellen Gener-ierung Multimodaler Dialogbeitra¨ge.* Sankt Augustin: infix.

Robinson, J. A. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *J.ACM*, 12:23–41.

Russel, Stuart J. and Shlomo Zilberstein. 1991. Composing Real-Time Systems. In *Proc. of the 12th IJCAI*, pages 212-217, Sidney, Australia, August.

Sadler, Louisa and Henry S. Thompson. 1991. Structural Non-Correspondence in Translation. In *Proc. of the 5 th EACL*, pages 293-298, Berlin, Germany.

Sampson, G. R. 1983. Context-Free Parsing and the Adequacy of Context-Free Languages. In M. King, editor, *Parsing Natural Language.* Academic Press, London, pages 151-170.

Samuel, Arthur G. 1990. Using Perceptual-Restoration Effects to Explore the Architecture of Perception. In G. Altmann, editor, *Cognitive Models of Speech Processing.* MIT Press, Cambridge, MA, chapter 14, pages 295-314.

Sato, S. andM. Nagao. 1990. Towards memory based translation. *InProc. of the 13 th COLING*, pages 3/247-3/252, Helsinki, Finland.

Satta, G. and Oliviero Stock. 1989. Formal Properties and Implementation of Bidirectional Charts. In *Proc. International Joint Conference on Artificial Intelligence*, pages 1480-1485, Detroit, MI.

Satta, Giorgio and Oliviero Stock. 1994. Bidirectional context-free grammar parsing for natural language processing. *Artificial Intelligence*, 69:123-164.

Schollhammer, Thomas. 1997. Ubersetzung von Idiomen in einer Speechumge-bung. Unveroff. Studienarbeit, Universitat Hamburg.

Schroder, Martin. 1993. *Erwartungsgestutzte Analyse medizinischer Befundungs-texte. Ein wissensbasiertes Modell zur Sprachverarbeitung.* Ph.D. thesis, Univ. of Hamburg, Hamburg.

Schubert, Klaus. 1992. Esperanto as an intermediate language for Machine Translation. In John Newton, editor, *Computers in Translation: A Practical Appraisal.* Routledge, London, pages 78-95.

Seligman, Mark, Christian Boitet and Boubaker Meddeb Hamrouni. 1998. Transforming Lattices into Non-deterministic Automata with Optional Null Arcs. In *COLING98P*, COLING98L.

Sheil, B. A. 1976. Observations on Context-Free Parsing. *Statistical Methods in Linguistics*, 6:71-109.

Shieber, Stuart M. 1984. The Design of a Computer Language for Linguistic Information. In*Proc. of the 10 th COLING*, pages 362-366, Stanford, CA, July.

Shieber, Stuart M. 1985. Evidence Against the Context-Freeness of Natural Languages. *Linguistics and Philosophy*, 8:362-366.

Shieber, StuartM., GertjanvanNoord, RobertC. Moore andFernando C.N. Pereira. 1989. A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms. *InProc. of the 27th ACL*, pages 7-17, Vancouver.

Shieber, Stuart M., Gertjan van Noord, Robert C. Moore and Fernando C.N. Pereira. 1990. Semantic-Head-Driven Generation. *Computational Linguistics*, 16(1):30-42.

Shillcock, Richard. 1990. Lexical Hypotheses in Continuous Speech. In G. Altmann, editor, *Cognitive Models of Speech Processing.* MIT Press, Cambridge, MA, chapter 2, pages 24-49.

Shillcock, Richard and Ellen Gurman Bard. 1993. Modularity and the Processing of Closed-class Words. In Gerry T. M. Altmann and Richard Shillcock, edi-

tors, *Cognitive Models of Speech Processing: The Second Sperlonga Meeting*. Lawrence Erlbaum, Hove, UK, chapter 9, pages 163-185. Sobashima, Yasuhiro, Osamu Furuse, Susumu Akamine, Jun Kawai and Hitoshi Iida. 1994. A Bidirectional, Transfer-Driven Machine Translation System for Spoken Dialogues. In *COLING-94: The 15th International Conference on Computational Linguistics*, pages 64-68, Kyoto, Japan. Somers, Harold L. 1993. Current Research in Machine Translation. *Machine Translation*, 7:231-246. Sommerville, Ian. 1996. *Software Engineering*. Addison-Wesley. Spilker, Jorg. 1995. Parallelisierung eines inkrementellen aktiven Chart-Parsers.

Verbmobil-Report 105, Universitat Erlangen-Nu¨rnberg. Steel, Sam and Anne de Roeck. 1987. Bidirectional Chart Parsing. In *Proc. of the 1987 AISB Conference*, pages 223-235, Edinburgh. Steele, G. L. and W. D. Hillis. 1986. Connection Machine Lisp: Fine-Grained Symbolic Processing. In *Proceedings of 1986 Symposium on Lisp and Functional Programming*, pages 279-297'. Steinbiß, V,B.H. TranandH. Ney. 1994. Improvements in Beam Search. *InProc. ICSLP-94*, pages 2143-2146, Yokohama, Japan. Stock, Oliviero. 1989. Parsing with Flexibility, Dynamic Strategies, and Idioms in Mind. *Computational Linguistics*, 15(1): 1-18, March. Stock, Oliviero, Rino Falcone and Patrizia Insinnamo. 1988. Island Parsing and Bidirectional Charts. *InProc. of the 12th COLING*, pages 636-641, Budapest, Hungary, August. Strom, Volker and G.Widera. 1996. What's in the "pure" Prosody? *InProc. ICSLP 1996*, Philadelphia, PA. Tanenhaus, M. K., J. M. LeimanandM. S. Seidenberg. 1979. Evidence for multiple stages in the processing of ambiguous words in syntactic contexts. *Journal of Verbal; Learning and Verbal Behavior*, 18:427-440. Thompson, Henry S. and Graeme Ritchie. 1984. Implementing Natural Language Parsers. In T. O'Shea and M. Einsenstadt, editors, *Artificial Intelligence — Tools, Techniques, and Application*. Harper and Row, London, pages 245-300. Tran, B. H., F. Seide and V. Steinbiss. 1996. A word graph based n-best search in continuous speech recognition. *In ICSLP*. Tropf, Herbert S. 1994. Spontansprachliche syntaktische Phanomene: Analyse eines Korpus aus der Domane "Terminabsprache". Technical report, Siemens AG, Mu¨nchen, January. Vitter, J. S. andR. A. Simons. 1986. New Classes for Parallel Complexity: A Study of Unification and Other Complete Problems for P. *IEEE Trans. Comp.*, pages C-35. Vogel, Carl, Ulrike Hahn and Holly Branigan. 1996. Cross-Serial Dependencies Are Not Hard to Process. *InProc. of the 16th COLING*, pages 157-162, Copenhagen, Denmark, August.

Wachsmuth, I. and Y. Cao. 1995. Interactive Graphics Design with Situated Agents. In W. Strasser and F. Wahl, editors, *Graphics and Robotics*. Springer Verlag, pages 73–85.

Wahlster, Wolfgang. 1993. Translation of Face-to-Face-Dialogs. In *Proc. MT Summit IV*, pages 127–135, Kobe, Japan.

Wahlster, Wolfgang. 1997. Verbmobil: Erkennung, Analyse, Transfer, generierung und Synthese von Spontansprache. Verbmobil-Report 198, DFKI, Saarbru¨cken.

Waibel, Alex. 1996. Interactive Translation of Conversational Speech. *Computer*, 29(7), July.

Waibel, Alex, A. M. Jain, A. E. McNair, H. Saito, A. G. Hauptmann and J. Tebel-skis. 1991. JANUS: A Speech-to-Speech Translation System Using Connec-tionist and Symbolic Processing Strategies. In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 1991*.

Wang, Ye-Yi and Alex Waibel. 1995. Connectionist Transfer in Machine Translation. In *Proc. International Conference on Recent Advantages in Natural Language Processing*, pages 37–44, Tzigov Chark, Bulgaria, September.

Warren, David. 1983. An Abstract Prolog Instruction Set. Technical Note 309, SRI International, Menlo Park, CA.

Warren, R. M. 1970. Perceptual restoration of missing speech sounds. *Science*, 167:392–393.

Weber, Hans. 1992. Chartparsing in ASL-Nord: Berichte zu den Arbeitspaketen P1 bis P9. Technical Report ASL-TR-28-92/UER, Universita¨t Erlangen-Nu¨rnberg, Erlangen, December.

Weber, Hans. 1995. *LR-inkrementelles, Probabilistisches Chartparsing von Wor-thypothesengraphen mit Unifikationsgrammatiken: Eine Enge Kopplung von Suche und Analyse*. Ph.D. thesis, Universita¨t Hamburg.

Weber, Hans H., Jan W. Amtrup and Jo¨rg Spilker. 1997. Innovative Systemar-chitekturen zur Inkrementellen Interaktiven Verarbeitung. *K u¨nstliche Intelli-genz*, 11(4):26–30, December.

Weber, Volker. Forthcoming. *Funktionales Konnektionistisches Unifikations-basiertes Parsing*. Ph.D. thesis, Univ. Hamburg.

Weisweber, Wilhelm. 1992. Term-Rewriting as a Basis for a Uniform Architecture in Machine Translation. In *COLING-92: The 15th International Conference on Computational Linguistics*, pages 777–783, Nantes, France.

Weisweber, Wilhelm. 1994. The experimental MT system of the project KIT FA S T. I n *Machine Translation: Ten Years On*, pages 12–1–12–19, Cranfield, UK, November.

Weisweber, Wilhelm and Christa Hauenschild. 1990. A Model of Multi-Level Transfer for Machine Translation and Its Partial Realization. KIT-Report 77, Technical University of Berlin.

Whitelock, P. 1992. Shake-and-Bake Translation. In *COLING-92: The 15th International Conference on Computational Linguistics*, pages 784–791, Nantes,

France.

Winograd, Terry. 1983. *Language as a Cognitive Process. Volume I: Syntax.* Reading, MA: Addison-Wesley.

Wintner, Shuly. 1997. *An Abstract Machine for Unification Grammars.* Ph.D. thesis, Technion - Israel Institute of Technology, Haifa, Israel, January.

Wintner, Shuly and Nissim Francez. 1995a. Abstract Machine for Typed Feature Structures. In *Proceedings of the 5th Workshop on Natural Language Understanding and Logic Programming*, Lisbon, Spain.

Wintner, Shuly and Nissim Francez. 1995b. Parsing with Typed Feature Structures. In *Proceedings of the 4th International Workshop on Parsing Technologies (IWPT95)*, pages 273-287, Prague, September. Charles University.

Wiren, Mats. 1988. On Control Strategies and Incrementality in Unification-Based Parsing. Linkoping Studies in Science and Technology, Thesis No. 140. Master's thesis, Linkoping University.

Wiren, Mats. 1992. *Studies in Incremental Natural-Language Analysis.* Ph.D. thesis, Linkoping University, Linkoping, Sweden.

Woodland, P. C, C. J. Leggetter, J. J. Odell, V Valtchev and S. J. Young. 1995. The 1994 HTK large vocabulary speech recognition system. In *ICASSP95.*

Woods, W A. 1973. An Experimental Parsing System for Transition Network Grammars. In Randall Rustin, editor, *Natural Language Processing.* Algorithmic Press, New York.

Worm, Karsten and C. J. Rupp. 1998. Towards Robust Understanding of Speech by Combination of Partial Analyses. In *Proc. of the 13th ECAI*, Brighton, UK.

Worm, Karsten L. 1998. A Model for Robust Processing of Spontaneous Speech by Integrating Viable Fragments. In *COLING98P*, COLING98L.

Wu, Dekai. 1995a. Grammarless Extraction of Phrasal Translation Examples From Parallel Texts. In *TMI95P*, TMI95L.

Wu, Dekai. 1995b. Stochastic inversion transduction grammars, with application to segmentation, bracketing, and alignment of parallel corpora. In *Proc. of the 14th IJCAI*, pages 1328-1335, Montreal, Canada, August.

Ying, H. G. 1996. Multiple Constraints on Processing Ambiguous Sentences: Evidence from Adult L2 Learners. *Language Learning*, 46(4):681-711, December.

Zajac, Remi. 1990. A Relational Approach to Translation. In *Proc. of the 3rd Int. Conf. on Theoretical and Methodological Issues of Machine Translation*, Austin, TX.

Zajac, Remi. 1991. Notes on the Typed Feature System. Technical report, IMS-CL, Stuttgart.

Zajac, Remi. 1992. Inheritance and Constraint-Based Grammar Formalisms. *Computational Linguistics*, 18(2):159–182.

Zajac, Remi. 1998. Feature Structures, Unification and Finite-State Transducers. In *FSMNLP'98, International Workshop on Finite State Methods in Natural Language Processing*, Ankara, Turkey, June.

Zajac, Remi, Marc Casper and Nigel Sharples. 1997. An Open Distributed Architecture for Reuse and Integration of Heterogeneous NLP Components. In *Proc.*

of the 5 th Conference on Applied Natural Language Processing, Washington, D.C. Zechner, Klaus and Alex Waibel. 1998. Using Chunk Based Partial Parsing of Spontaneous Speech in Unrestricted Domains for Reducing Word Error Rate in Speech Recognition. In *COLING98P*, COLING98L. Zwitserlood, P. 1989. The Locus of Effects of Sentential-Semantic Context in Spoken-Word Processing. *Cognition*, 32:25-64.

Glossary

Abbreviations and Acronyms

ALE	Attribute Logic Engine	76
ASL	Architecture of Speech-Language Systems	17
ATIS	Air travel information System	22
ATN	Augmented Transition Network	65
CFG	Context-Free Grammar	66
CKY	Cocke, Kasami, Younger algorithm	43
CSP	Communicating Sequential Processes	96
DAG	Directed acyclic graph	29
DCG	Definite Clause Grammar	67
EVS	Ear-voice span	15
FUG	Functional Unification Grammar	73
HMM	Hidden Markov Model	30
HPSG	Head Driven Phrase Structure Grammar	66
ICE	INTARC Communication Environment	96
IDL	INTARC Data Layer	98
ILS	INTARC License Server	98
INTARC	INTeractive ARChitecture	23
LFG	Lexical Functional Grammar	67
LR	Left-Right (incrementality)	5
MILC	Machine Interpreting with Layered Charts	1
MT	Machine Translation	128
MUC	Message Understanding Conference	110
NLP	Natural Language Processing	3
NP	Noun Phrase	117
PVM	Parallel Virtual Machine	97
RPC	Remote Procedure Call	96
SLT	Spoken Language Translator	22
SSSP	Single Source Shortest Paths	60
TDMT	Transfer-Driven Machine Translation	22
TFS	Typed Feature Structure	74
UTAG	Unification Tree Adjoining Grammar	67
VIT	Verbmobil Interface Term	77
XDR	eXternal Data Representation	98

Symbols

T	The most general type in a type lattice	68
±	The inconsistent type in a type lattice	68
->	Adjacency relation	27
A	Reachability relation	27
zi	Subsumption relation	69
n	Unification	69
#m()	In-degree of a vertex	3
#o«t()	Out-degree of a vertex	32
!SIL,	Silence label	48
a	Access function for the start vertex of an edge	27
/?	Access function for the end vertex of an edge	27
<5(v,v')	Distance between two vertices	28
£	Set of Edges	25
G	Graph	25
G(0,0)	The empty graph	25
He	Hyperedge associated with a word graph edge	55
£	Set of labels	26
I	Access function for the label of an edge	27
p(G)	Number of paths in a graph	38
TDC	Type Definition Language	78
T	Topological order of a graph	34
V	Set of vertices	25
VM	Root of a graph	32
(/\	Final vertex of a graph	32
W	Set of weights	26
w	Access function for the weight of an edge	27

Index