Lehrstuhl für Mensch-Maschine-Kommunikation
Technische Universität München

Efficient Integration of Hierarchical Knowledge Sources and the Estimation of Semantic Confidences for Automatic Speech Interpretation

Robert Lieb

Vollständiger Abdruck der
von der Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität München
zur Erlangung des akademischen Grades
eines **Doktor-Ingenieurs** genehmigten Dissertation.

Vorsitzender: **Univ.-Prof. Dr.-Ing. Georg Färber**

Prüfer der Dissertation:

1. **apl. Prof. Dr.-Ing., Dr.-Ing. habil. Günther Ruske**
2. **Univ-Prof. Dr.-Ing. Gernot A. Fink** (Universität Dortmund)

Die Dissertation wurde am **19.06.2006** bei der
Technischen Universität München eingereicht und durch die
Fakultät für Elektrotechnik und Informationstechnik
am **03.11.2006** angenommen.

Abstract

This thesis presents a system for the interpretation of natural speech which serves as input module for a spoken dialog system. It carries out the task of extracting application-specific pieces of information from the user utterance in order to pass them to the control module of the dialog system.

By following the approach of integrating speech recognition and speech interpretation, the system is able to determine the spoken word sequence together with the hierarchical utterance structure that is necessary for the extraction of information directly from the recorded speech signal.

The efficient implementation of the underlying decoder is based on the powerful tool of weighted finite state transducers (WFSTs). This tool allows to compile all involved knowledge sources into an optimized network representation of the search space which is constructed dynamically during the ongoing decoding process.

In addition to the best-matching result, the integrated decoder architecture allows to determine grammatical alternatives which are exploited to estimate semantic confidence values for the extracted pieces of information. This new method improves the robustness against interpretation errors without requiring any additional knowledge source.

Zusammenfassung

Diese Arbeit beschreibt ein System zur Interpretation von natürlicher Sprache, das als Teil eines automatischen Dialogsystems applikations-spezifische Informationen aus Benutzeräußerungen extrahiert. Durch die Vereinigung von Spracherkennung und -interpretation gelingt es, die für die Informationsextraktion erforderliche hierarchische Struktur einer Äußerung direkt aus dem Sprachsignal zu gewinnen.

Die effiziente Realisierung des Dekoders beruht auf dem mächtigen Kalkül der gewichteten endlichen Transduktoren (engl. WFST), der voranschreitend mit dem Ablauf des Dekodiervorgangs aus allen involvierten Wissensquellen eine optimale Netzwerkdarstellung des aktiven Suchraums generiert.

Neben dem besten Ergebnis erlaubt die integrierte Dekoderarchitektur die Erzeugung von grammatischen Alternativen, auf deren Basis semantische Konfidenzen für die extrahierten Informationen geschätzt werden. Damit wird die Fehlerrobustheit erhöht, ohne dass hierfür eine weitere Wissensquelle erforderlich ist.

Danksagungen

Ich möchte mich herzlich bei meinem ehemaligen Kollegen Matthias Thomae für die sehr gute Zusammenarbeit während unserer gemeinsamen Forschungstätigkeit am Lehrstuhl für Mensch-Maschine-Kommunikation bedanken.

Dank gebührt natürlich ebenso Prof. Ruske, welcher mich während dieser Zeit bestens betreut hat; seine Bürotüre stand stets offen für Fragen und Diskussionen. Vielen Dank auch an den Zweitgutachter Prof. Fink, an den Lehrstuhlinhaber Prof. Rigoll, sowie an die BMW AG, welche durch die Förderung von Forschungs-Projekten diese Arbeit finanziert und somit möglich gemacht hat.

“Last but not least” möchte ich mich bei meiner Frau Ofelia bedanken, die mir auch in schwierigen Phasen der Doktorarbeit stets den Rücken freigehalten hat.

Robert Lieb
München, Januar 2007

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Spoken language is the most natural and direct means of interpersonal communication and one of the most important skills that make up the human intellect. Therefore, it is not astonishing that the vision of artificial intelligence paints the picture of machines that have the ability to communicate with humans by spoken language. Almost everybody knows the science fiction scenarios where intelligent machines with conversational skills, like androids or space ships, play the role of omniscient counselors which help humans to solve complex problems or to make difficult decisions.

The belief that those visions will once come true has been nourished by the overwhelming development of the computer technology that brought us the so-called "Age of Information". It is actually true that our daily interaction with information processing systems provides many reasonable usage scenarios for the communication between man and machine by spoken language. Spoken language is particularly interesting in usage scenarios where conventional input and output devices like keyboards or displays are not available, or can be provided only in a reduced form. This may be the case due to the limited size of the corresponding electronic device or because the user's tactile and visual communication channels are reserved for other tasks. Examples for such usage scenarios are mobile phones or personal digital assistants, onboard-computers in automobiles and voice portals. Typical applications provided in such environments are the management of the user's personal communication (e.g. phone calls, messages, addresses and phone numbers), the handling of the large amount of functions available in vehicle cockpits (e.g. the control of the car's navigation and entertainment systems), public information services (e.g. news, weather, train or flight schedules) and customer services (e.g. product information, online-shopping).

In comparison to other communication modalities, spoken language has the appealing advantage that it combines intuitive usage with universal expressiveness. On the one hand the interaction with spoken language does not require the handling of some complex input device, which may distract the user from other tasks. On the other hand, almost everything can be expressed by spoken language, such that there is almost no restriction concerning the semantic complexity of the transported information. There are only a few examples, like the adjustment of continuous parameters (e.g. the volume of the car radio), where spoken language is less appropriate.

While there is no doubt about the great potential of spoken language for human-machine communication, the realization of technical systems that actually tap this potential remains an open challenge which is still subject of intensive scientific research. The encountered difficulties are above all the consequence of the fact that when using our natural way to communicate we usually tend to assume a dialog partner with human intellect that can only be simulated to a very limited extend by means of today's computing machines. At the same time, many of the above mentioned applications actually aim on replacing humans in situations where they are not available or in order to save payroll costs.

In contrast to science-fiction scenarios where the final barrier between man and machine is often drawn with respect to human emotions, in reality difficulties arise on a much lower level. One of the main problems is to find an adequate knowledge representation and a corresponding formalism that in the end allows to determine the appropriate system reaction. A plethora of different knowledge representation and processing formalisms has been suggested so far. Unfortunately, all of them turned out to be brittle when attempting to create a general ontology, that captures the whole world's knowledge in order to provide a basis for the simulation of domain-independent conversational skills. Furthermore, the use of spoken language additionally entails the problem of processing the speech signal which represents the contained information in a non-symbolic way. On the input side, this affords the heavy reduction and normalization of the observed signal data in order to extract the transported high-level information. For the output of spoken language just the opposite problem has to be solved, namely the diversification and individualization of the high-level information in order to achieve a natural wording and voice.

A pragmatic way to tackle the above mentioned problems is to give up the ambitious goal to find an universal approach to the simulation of domain-independent conversational skills. Therefore, without worrying too much about the lack of a general theory, engineers are developing spoken dialog systems that are tailored for specific application domains. Spoken dialog systems are based on the interconnection of software modules which are responsible for speech recognition and understanding, dialog and database management, as well as response generation and speech synthesis. These modules usually exploit separate knowledge sources and exchange information over a specific communication protocol. The interaction with the user usually takes place in the following manner: the dialog is initiated with a system prompt that greets the user. Then, the system waits for the user utterance, which is processed by the speech recognition and understanding module in order to extract the application-specific information which is passed to the module responsible for the dialog management. By processing the extracted information, the dialog management determines the appropriate system reaction which is then translated into an adequate wording that is finally passed to the speech synthesis module which plays the system response to the user. During the ongoing conversation which consists of a sequence of such dialog cycles the dialog system has to cope with the demanding task to extract the relevant information from consecutive user utterances and to translate this knowledge into a cooperative policy that supports the user in accomplishing the intended actions.

In practice, spoken dialog systems are often integrated into multi-modal user interfaces that allow the use of spoken language in combination with haptic input devices and displays. In many cases this leaded to solutions which provide rather a voice control of the conventional human-machine interface than a real conversational interface. Typical for those kind of systems is a small vocabulary that only allows utterances in a command-like style. In the case of complex actions that require the input of several pieces of information, the dialog is heavily directed by the system which asks the user in a predefined order to provide each piece of information in a separate utterance. Voice interfaces with these restricted capabilities have the advantage of an easy-to-manage configuration that can be rapidly developed and tested in a consistent way and thus represent the state-of-the-art in today's commercial solutions. However, they achieve only a moderate acceptance due to the simple reason that for many users the use of spoken language implies conversational skills that the system does not provide. The gap between their own conversational competence and the restricted system capabilities makes those users feel uncomfortable such that they prefer conventional interaction modalities which might be more cumbersome to use but provide a handling that is clearly defined.

Therefore, a major goal of the ongoing scientific research is to equip spoken dialog systems with more powerful conversational capabilities, which in particular are natural language understanding and a mixed-initiative dialog management. The latter allows the user to decide freely about the complexity of his utterances. This means, independently of the current dialog context, the user has the choice either to use a complex utterance which contains several pieces of information or to give only a part of the necessary information. In this case the system has to answer with corresponding queries that ask the user to provide the missing pieces of information, which can even be the subject of the request itself. Furthermore, the user may reject system responses in order correct himself or interpretation errors committed by the system. An important prerequisite for the application of dialog management techniques that provide such advanced conversational capabilities is the processing of complex utterances in natural language, in order to extract the contained application-specific information. The task of spoken language understanding is the central subject of the present work and will be introduced in the following section.

At the end of this general introduction to the background of the present work, it should be mentioned that it is still questionable whether the just mentioned conversational skills are actually realizable with reasonable effort, even when the dialog is restricted to specific application domains. The reason for this is frightening simple and typical for large software projects (what spoken dialog systems finally are): the more complex a system gets, the more unmanageable becomes its administration which in the context of a spoken dialog systems means that it becomes impracticable to eliminate causes of erroneous system behavior due to the complex network of dependencies that make up such systems. It remains to be seen whether in the near future a compromise between the offered conversational skills and their technical feasibility can be established, that is widely accepted by the users. When focusing only on the effort to approach the ideal of human conversational skills, this goal may become difficult to reach. On the other hand, it helps a lot to give conversational interfaces a uniform design which makes the user change his mental model of a talking machine – away from the idea of a human dialog partner in direction to what the user is actually talking to, namely a technical device which requires a specific mode of verbal communication.

1.1 Spoken language understanding

The present thesis focuses on the task of extracting application-specific information from the recorded speech signal of user utterances in natural language. Linguists, who consider the underlying problem from a more general point of view, have divided the involved knowledge into several conceptual levels (see [All95]):

- **Acoustic-phonetic level.** The acoustic-phonetic level describes the connection between the acoustic speech signal and a symbolic representation of the pronunciation of words or parts of it.
- **Morphological level.** The morphological level considers the formation of words from orthographical units (e.g. inflections, compound words, etc.)
- **Syntactic level.** The syntactic level deals with the construction of valid sentences that follow specific grammatical rules.
- **Semantic level.** The semantic level considers the meaning of words and phrase segments independently of the context of a particular sentence.
- **Pragmatic level.** The pragmatic level deals with knowledge which is not explicitly expressed by a particular phrase but is necessary for its successful interpretation ("world" knowledge).
- **Discourse level.** The discourse level considers how preceding sentences influences the interpretation of the next sentence.

The linguistic knowledge classification scheme provides a starting point to identify subtasks that can be solved independently. However, when going into the details it rapidly turns out that it is very difficult to isolate separate knowledge representations, due to the fact that the conceptual levels don't represent a strict hierarchy but tend to intersect each other.

The most obvious intermediate knowledge representation is the orthographic transcription of a particular user utterance, which consists of a sequence of words. In consequence, this representation links two different scientific communities, namely the speech recognition community, which deals with digital signal processing and pattern recognition, and the community of computational linguistics which addresses the problem of natural language understanding. The goal of speech recognition is to find out the correct word sequence from the speech signal of the user utterance, whereas the goal of natural language understanding is to extract the semantic content which is transported by the wording of the user utterance. These two tasks have quite different prerequisites: with the word sequence, speech recognition has a well-defined output, but entails the problem of processing the noisy speech signal, which shows great variance from various points of view, like for example the pronunciation of words or the acoustic characteristics of different speakers and recording conditions. To the contrary, natural language understanding has a well-defined input, whereas the subtle outcome "semantic content" is so difficult to grasp, that no widely accepted canonical representation could be established, so far. Moreover, speech recognition can be seen as a segmentation task, which reveals the mapping between consecutive sections of the speech signal and the corresponding words, whereas the task of natural language understanding generally cannot be considered on the basis of such a simple relation between input and output.

1.1.1 Speech recognition

Currently, the by far most successful approach to tackle the task of speech recognition is by means of statistical models which describe the variety of acoustical patterns that may occur for each unit of a phonetic alphabet (see for example [Rab89]). This alphabet provides all phonetic units which are necessary to put together the pronunciation of arbitrary words from a particular language. The acoustic-phonetic models for the pronunciation of words are combined with a so-called "language model", which describes the variety of possible word sequences usually in form of a prior distribution. A speech recognition system compiles all those knowledge sources into an integrated search space and uses an efficient decoding algorithm to determine the predicted word sequence which best matches the observed speech signal. The straightforwardly structured knowledge representation (word sequence, word, phonetic unit and corresponding acoustic pattern) permits a very successful application of data-driven training algorithms which allow to estimate the large amount of model parameters. Provided comprehensive data collections, speaker-invariant acoustic models can be built almost automatically from the recorded speech signal data and the corresponding orthographic corpus transcription.

1.1.2 Natural language understanding

In contrast to the case of speech recognition, the field of natural language understanding did not experience a comparable breakthrough of a singular problem solving approach. This is above all the consequence of the difficulty of defining the task of natural language understanding itself. From the linguistic point of view, this definition implies an overwhelming amount of complex knowledge involving all higher conceptual levels of the just discussed knowledge classification scheme (morphological, syntactic, semantic, pragmatic and discourse level). As already mentioned, the currently known representation methods and processing formalisms do not allow such a broad approach that considers all this knowledge in general. Therefore, in the narrow context of spoken dialog systems, the understanding task is usually defined as the extraction of application-specific information. The crucial question how to process this information in order to determine an appropriate system reaction is not included in this problem definition and is simply declared to be part of the dialog management, which receives the extracted information. Because the term "understanding" may nevertheless imply some kind of system reaction, in the following the less ambitious term "semantic interpretation" will be preferred instead.

Despite of being limited to a specific application domain, the problem of representing and processing the knowledge which is necessary to extract application-specific information from the wording of possible user utterances still remains difficult to grasp. In fact, it is even hard to give a unique definition for the "application-specific information" itself, which heavily depends on the nature of the approach which further processes the extracted information. In consequence to this great degree of freedom concerning the actual problem definition, in scientific literature one can find a multitude of approaches to domain-specific semantic interpretation which are difficult to compare. In general, one can distinguish between approaches that rely on rule-based knowledge that is handcrafted following linguistically motivated guidelines (e.g. MIT's TINA and SRI's Gemini systems, see [Sen92, DG93]) and approaches which favor the application of machine-learning techniques similar to those which have been applied with great success in the discipline of speech recognition (e.g. AT&T's Chronous and BBN's HUM systems, see [PT92, MB95]). Unfortunately, this crude classification has often been overstressed, in order to evoke a superficial discussion about the question whether machine-learning approaches are more adequate than rule-based approaches. The truth however is, that in the context of semantic interpretation, the application of machine-learning techniques still requires a considerable amount of expert knowledge which provides the structural framework for the training of statistical parameters, or which guides the translation of the captured application-specific information into its final form. Therefore, the actual question concerns the adequate design of the handcrafted knowledge which allows a reasonable application of machine-learning techniques in order to infer at least some knowledge from data, in particular those which is hard to specify for a human expert. Until today, there are ongoing research activities in order to find consistent solutions for this difficult problem, most often by incorporating knowledge representations which abandon important linguistic principles, like a clear distinction between syntax and semantics, whereas purely linguistically motivated representations are more and more considered as old-fashioned (see for example [HY05]).

Unfortunately, in practice, the size of domain-specific user utterance collections is often limited due to the effort to conduct the necessary experiments, which reduces the impact of sophisticated machine-learning techniques. Since this has been the case in the context of the present work, this thesis adopts a standard approach to semantic interpretation, which is based on handcrafted semantic-syntactic knowledge in form of a so-called "semantic grammar". This approach is widely used in practical applications and part of an upcoming industry standard (see [W3C04]). Semantic grammars incorporate context-free grammar rules that associate the wording of possible user utterances with a domain-specific hierarchical structure, that guides the intended extraction of application-specific information. This information is represented in form of "slot-value-pairs" (semantic variables), which provides a straightforward interface to the dialog management. The extraction of slot-value pairs however involves additional rules that are executed during the structural analysis that is effected by the aid of a parser that applies the given semantic grammar to the wording of a specific user utterance.

1.1.3 Coupling speech recognition and interpretation

This thesis addresses above all the processing of spoken language, which requires the coupling of speech recognition and semantic interpretation. The recognized word sequence provides a simple interface for a serial coupling of the standard tools speech recognizer and parser, which have emerged separately in the scientific communities of speech recognition and natural language processing. However, the serial or "loose" coupling has been criticized a lot, because of the fact that the knowledge sources of speech recognizer and parser contain redundant information and cannot be applied simultaneously. For the task to determine the semantic interpretation that best matches the recorded speech signal, this approach is definitely suboptimal and may even lead to unrecoverable errors in the case of inconsistencies in the redundant parts of the involved knowledge sources. This happens if the language model predicts word sequences that are not covered by the parser grammar, or vice versa (especially stochastic language models incorporate constraints which do not match with the rule-based structure of a semantic grammar).

Nevertheless, because of its straightforward realization, the sequential coupling of recognizer and parser is common practice. In addition, it has been suggested to pass a whole bunch of possible word sequences to the parser in order to have a backup for the case that the best-matching word sequence is not included in the parser grammar (see [CRAR99, HW01]), respectively to utilize robust parsing techniques which allow to skip words which don't fit into the parser grammar (see [Abn96]). These precautions allow to weaken the just mentioned drawbacks and even provide some robustness against interpretation errors. However, the usage of redundant and partially inconsistent knowledge sources hamper the practical administration of the speech interpretation system with respect to the intended application as a part of a spoken dialog system (for example when having to eliminate specific interpretation errors or when having to exchange parts of the knowledge sources). Therefore, it has been suggested to harmonize the knowledge sources of speech recognizer and parser, for example by utilizing the parser grammar itself as language model. This is the method of choice in several commercial solutions for speech interpretation and involves the compilation of the parser grammar into a word-based language model, that can be processed by the speech recognizer (see for example [Nua01, HH00]). Unfortunately, this solution still has the disadvantage of the sequential application of speech recognizer and parser, which may cause a delay in the system reaction, if the parser cannot determine the result in a short amount of time. Another drawback is the initialization effort for the compilation of the semantic grammar, which has to be repeated when exchanging parts of the semantic grammar during the ongoing dialog.

These circumstances motivate the actual subject of this thesis, which is the integration of speech recognition and semantic interpretation by a an approach that tightly couples acoustic phonetic decoding and parsing and therefore applies all involved knowledge sources simultaneously (see section 1.3).

1.2 Issue of robustness

In practice, it cannot be avoided that user utterances may contain parts which are not included in the limited scope of the semantic-syntactic knowledge source. After all, when suggesting natural speech input, one cannot expect from a user that he knows exactly which particular wording is covered and which not. Furthermore, even if the wording is theoretically part of the semantic-syntactic knowledge source, there may still occur an interpretation error due to imperfect models on the acoustic-phonetic level.

The problem of uncovered utterance parts becomes especially critical when assuming a tight integration of speech recognition and semantic interpretation. Without further precautions, there is no certitude about whether an extracted piece of information relies on a good or a bad match between the integrated knowledge sources and the observed speech signal. The knowledge about potential interpretation errors, respectively their prevention, is however very important for the further processing of the extracted information inside the dialog management which plans the following system reactions. During the corresponding decision process, the dialog management should not rely on unsafely extracted information that may cause a misdirected dialog course which confuses the user. Potential interpretation errors should either be discarded or exploited to trigger a dialog step which asks the user to clarify his last statement. At the same time, the dialog management should not pursue an overcautious strategy which asks the user to confirm pieces of information that actually have been extracted correctly.

This thesis will present methods to improve the robustness against interpretation errors, particularly by taking advantage of the pursued integrated speech interpretation approach. These methods are based on two popular approaches that allow to prevent recognition errors in word-based speech recognition systems. The first of these approaches consists in integrating a general model for unknown words into the language model, such that these words can be identified immediately by the recognition process. However, it cannot be prevented that the general model for unknown words, which is also called "out-of-vocabulary" model, conflicts with the acoustic-phonetic modeling that is provided for the explicitly known words. Therefore, in practice this allows only a compromise which tries to identify as much unknown words as possible, however without committing too many errors on known words, which would be recognized correctly without the integration of the out-of-vocabulary model (see [Baz02]). The same underlying method of integrating explicit models is also applied to cope with disfluencies occurring in natural speech, like for example pauses, laughter, coughing, hesitations or cut-off words.

In contrast the use of explicit models that capture uninterpretable parts of the user utterance directly, the other approach for gaining robustness relies on an implicit estimation of confidence values which measure the reliability of each word in the recognized word sequence. A popular method effects this estimation on a network representing probable alternatives to the recognized word sequence which can be created by the integrated decoding algorithm without the need of any additional knowledge source (see [WSMN01]). Similar to the case of the out-of-vocabulary modeling approach, the confidence estimation is only an approximation which generally does not allow to identify all recognition errors without committing some errors on correctly recognized words, which implies a trade-off between errors due to the false acceptance or the false rejection of particular words.

1.3 Thesis contribution

Before presenting the detailed objective of this thesis, it is worthwhile to give a brief overview about the particular environment in which the actual scientific work took place. The framework of this work was provided by the industry-funded research project NaDia ("Natürlichsprachliche Dialogführung für die Nutzung komplexer Informationsdienste im Automobil"), which investigated the application of a multi-modal dialog system with natural speech input in order to provide an intuitive user interface for information services offered by the on-board computer of future automobiles.

The prerequisites of the project concerning the natural speech input were the following: In order to keep things manageable, it has been decided to utilize the already mentioned standard approach to semantic interpretation via the use of handcrafted semantic grammars which include context-free grammar rules as well as additional rules controlling the extraction of slot-value pairs that can be passed to the dialog management. Nevertheless, the goal was to develop a mixed-initiative dialog system, that has to cope with user utterances containing several pieces of information. In consequence, the manual design of the necessary semantic grammar becomes a pretty demanding task. The responsible expert could however take advantage of a corpus of user utterances that was collected during several experiments that invited test subjects to interact with a simulated dialog system. The exemplary application domain was an airport information service that provides various information about arriving and departing flights.

Provided these prerequisites, the major objective was to devise an efficient approach to speech interpretation that tightly integrates speech recognition and parsing and incorporates methods which improve the robustness against interpretation errors. Another goal was to find a reasonable way to augment the semantic grammar by stochastic weights, which can be estimated by the aid of the relatively small-sized data collection. The approach to automatic speech interpretation which is presented in this thesis meets these requirements in the following way: Instead of grammar rules, it utilizes the equivalent representation of the semantic grammar in form of a hierarchy of transition networks. The network representation of the semantic grammar allows a seamless integration of the other knowledge sources necessary for speech recognition (acoustic models and phonetic lexicon), which has the great advantage of retaining the basic decoding principle that is commonly used in word-based speech recognition systems. In contrast to a precompilation of the entire semantic grammar into a flat word-based language model, the presented decoding method requires only to precompile each grammar rule individually, which explicitly preserves the hierarchical structure of the underlying semantic grammar. Hence, the software tool which implements the decoding algorithm is called "one-stage decoder", because it time-synchronously decodes the best-matching parse tree directly from the recorded speech signal. The extraction of the application-specific information is effected in a post-processing step which applies the semantic grammar along the decoded parse tree and carries out the embedded rules for extracting the contained slot-value pairs which are finally passed to the dialog management. Furthermore, the network representation of the semantic grammar allows a straightforward incorporation of stochastic weights. These are determined on the collection of domain-specific user utterances which initially served as an aid during the manual design of the semantic grammar. The developed semantic grammar is then again applied to generate a parse tree annotation of the corpus, which allows to determine the stochastic weighting for each transition network individually. The resulting hierarchy of weighted transition networks is called "hierarchical language model".

In order to improve the robustness against interpretation errors, the thesis considers the above mentioned approaches, namely the explicit modeling of out-of-vocabulary words and the estimation of confidences, which are both refined in a way that they fit into the one-stage decoding framework. Concerning the out-of-vocabulary modeling method, this refinement consists in the integration of pronunciation models for out-of-vocabulary words and disfluencies of natural speech into the hierarchical language model, which allows an immediate "recognition" of uninterpretable utterance parts. Concerning the estimation of confidences, this thesis presents a novel method for estimating semantic confidences that measure the reliability of particular nodes of a decoded parse tree. These confidences are translated into confidences for the extracted slot-value pairs which can be exploited by the dialog management in order to decide how to proceed with the extracted information. The estimation of the parse tree node confidences is effected on a compact representation of grammatical alternatives to the best-matching parse tree, which is generated during the integrated decoding process.

Last but not least, it is important to point out that the present thesis touches substantial parts of the work of Thomae (see [Tho06]). This is the consequence of the intensive collaboration of the involved authors that generated a lot of synergies which have been exploited on both sides. Nevertheless, the authors agreed on a different thematic orientation of each thesis: Thomae's thesis focuses on the creation of the hierarchical language model including out-of-vocabulary models, whereas this thesis addresses the one-stage decoder itself, in particular its efficient realization, the generation of grammatical alternatives and their application for the estimation of semantic confidences. For the sake of completeness, Thomae's thesis already includes a conceptional description of a possible one-stage decoder architecture, which however implies a static search space organization that prevents an efficient integration of the involved knowledge sources. This thesis presents a more sophisticated architecture that allows the dynamic compilation of the search space which is effected during the ongoing decoding process. The computational overhead of the on-demand construction is compensated by the application of the sophisticated calculus defined on weighted finite state transducers (WFSTs) which allows the global optimization of the search space topology and therefore achieves a significant speed-up of the overall decoding process. Another important aspect is that the present thesis considers the actual information extraction (in form of slot-value pairs), whereas Thomae leaves this question open and evaluates the speech interpretation performance only on the basis of the decoded parse trees. On the other hand, details about the adjustment of the stochastic weights inside the hierarchical language model as well as details about the generation and the integration of the out-of-vocabulary models can be found in [Tho06].

The present thesis has the following structure: In order to prepare the reader for the intended integration of speech recognition and parsing, Chapter 2 provides the necessary theoretical background concerning the techniques which are commonly used in word-based speech recognition. Chapter 3 begins with an excerpt of formal language theory that leads straightforwardly to the discussion of the utilized approach to semantic interpretation by semantic grammars. After considering different strategies to couple speech recognition and parsing, the pursued one-stage decoding approach which relies on the network representation of the semantic grammar is introduced. The remainder of the chapter explains the creation process of the hierarchical language model, which represents the semantic grammar in form of a hierarchy of transition networks and contains stochastic weights which are estimated on the parse tree annotation of the domain-specific utterance collection.

Chapter 4 contains the detailed presentation of the one-stage decoder architecture and begins with the description of the decoder version that uses the already mentioned static search space organization. Before discussing the decoder architecture incorporating the dynamic search space compilation, the chapter provides an introduction to the necessary theoretical background, which is the theory of weighted finite state transducers (WFSTs). Then, it will be explained how the involved knowledge sources (hierarchical language model, pronunciation lexicon and acoustic models) are represented by WFSTs in order to apply automata operations that efficiently compose and optimize the search space. The decoding algorithm which uses the on-demand compilation of the search space will be presented in pseudo-code. The chapter is concluded by a performance comparison showing the improvement in efficiency that has been achieved by the dynamic search space compilation and optimization.

The first part of Chapter 5 shows how the commonly used lattice representation for word-based alternatives has been extended to provide a compact representation for grammatical alternatives in the form of a hierarchy of lattices. After providing the corresponding definition, it will be explained in which way the lattice hierarchy is generated from bookkeeping information that is recorded during the integrated decoding process. Furthermore, there is a section about the generation of a ranked list of parse trees, which can be used to generate alternatives on the basis of the extracted slot-value pairs. The second part of Chapter 5 explains the enhancement of a word-based confidence estimation method which uses the generated grammatical alternatives in order to estimate semantic confidences for each node of a decoded parse tree. The last section of the chapter presents the rule-based policy that associates the estimated parse tree node confidences with the extracted slot-value pairs.

Chapter 6 presents the employed methods for measuring the speech interpretation performance and discusses the obtained experimental results. The interpretation performance is considered on the basis of the decoded parse trees as well as on the basis of the extracted information. Because the subject of dialog management is beyond the scope of this work, the evaluation will include only off-line experiments that are carried out on a test subset of the domain-specific utterance collection. The conducted experiments have the main purpose to assess the quality of the confidence estimation, in particular in comparison to the method which explicitly models uninterpretable utterance parts by using out-of-vocabulary models. Furthermore, the chapter provides details about the domain-specific data collection (airport information corpus) and presents the setup of the one-stage decoder which has been used during the experiments. Finally, Chapter 7 summarizes the most important aspects of this thesis and provides an outlook to future work.

Chapter 2

Word-based speech recognition

This chapter is intended to give an introduction to word-based speech recognition, which is the task of determining the particular word sequence from a variety of predicted ones, that best matches the recorded speech signal. The standard acoustic modeling approach with the aid of “Hidden Markov Models” will be explained, as well as the “Viterbi” decoding algorithm that is used for recognition.

The techniques presented in this chapter will provide the basis for the main subject of this thesis, which is the tight integration of speech recognition and semantic interpretation (see chapters 3 and 4).

2.1 Decoding problem for speech recognition

The most common way to formulate the decoding problem for continuous speech recognition is the famous maximum likelihood approach [BJM83]: If X denotes the observed sequence of acoustic feature vectors extracted from the preprocessed speech signal[1] and W possible word sequences, then the decoding problem is to find the word sequence W^* that has the maximum posterior probability

$$W^* = \arg\max_W P\left(W|X\right). \tag{2.1}$$

Unfortunately, it's infeasible to estimate the posterior probabilities $P\left(W|X\right)$ directly. This is why equation 2.1 is transformed by the aid of Bayes' law to

$$W^* = \arg\max_W \frac{P\left(W\right)P\left(X|W\right)}{P\left(X\right)} = \arg\max_W P\left(W\right)P\left(X|W\right), \tag{2.2}$$

neglecting the term $P\left(X\right)$ which is not relevant for the maximization. Thus, another equivalent view onto the decoding problem is the maximization of the joint probability $P\left(X,W\right) = P\left(W\right)P\left(X|W\right)$. The factor $P\left(W\right)$ is the prior distribution over all possible word sequences, which is called language model. The other factor $P\left(X|W\right)$ associates possible word sequences with the acoustic feature vector sequence extracted from the recorded speech signal and therefore is called acoustic model. The important difference to the posterior probability $P\left(W|X\right)$ is the exchange of W and X in the probability's condition: Because $P\left(X|W\right)$ is conditioned

[1]For a detailed description of the standard preprocessing techniques which are applied in the context of this thesis, see [YEK$^+$02].

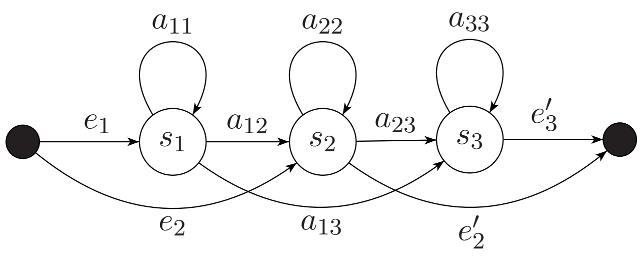

Figure 2.1: Example for a continuous HMM used for speech processing, represented by the parameter set $\lambda = (p(\mathbf{x}|s_i), a_{ij}, e_i, e'_i)$.

on discrete symbol sequences and not on the highly-dimensional and continuous space of possible feature vector sequences, the estimation for the acoustic model's probability distribution becomes a feasible task.

Equation 2.2 only poses the decoding problem and doesn't represent its solution. Nevertheless, it reveals some important concepts of possible implementations:

- The language model represents the space of possible recognition results and assigns a specific probability to every possible word sequence.
- The acoustic model assigns a specific probability for every possible word sequence considering the observed feature vector sequence.
- Both models have to be used simultaneously during the decoding process, which maximizes the product of language and acoustic model probability.

However, equation 2.2 doesn't give an answer to the following crucial problems:

- The acoustic model's probability distribution $P\left(X|W\right)$, referring to *sequences* of feature vectors and words, is still too complex to be specified directly. How can the acoustic model be split up to allow the processing of consecutive feature vectors, respectively words?
- The maximization of the product of language and acoustic model probability is a complex optimization task, that cannot be solved by a brute-force approach, evaluating all possible solutions. Which algorithms solve this problem efficiently?

Both questions lead to the use of Hidden Markov Models, which are a well-known tool among scientists dealing with speech recognition. For the sake of clarity, Hidden Markov Models are briefly revised in the following section; for a more detailed introduction to this subject see [Rab89] and also [Rus94].

2.2 Acoustic modeling by Hidden Markov Models

In the previous section it has been suggested to split up the complex acoustic model $P\left(X|W\right)$ into a set of models, which on the one hand correspond to the language model and on the other hand are able to process feature vectors consecutively, one after another. This is exactly what can be achieved by Hidden Markov Models. Because words are the symbolic units of the language model, it is straightforward to have as many acoustic models as there are different words in the language model. Every word can be represented by a Hidden Markov Model (HMM), which is a state transition model. It consists of a specific number of interconnected states s_i, like the example shown in figure 2.1. Regarding the intended processing of the acoustic feature vector sequence X, the HMM's parameter set λ contains the probability density functions $p(\mathbf{x}|s_i)$, which represent the probability that a particular feature vector $\mathbf{x}$ is emitted in state s_i. The term "emitting" points out that the HMM has to be regarded as a model that determines the probability with which it can generate the observed feature vector sequence. This model definition is also called continuous HMM, in contrast to discrete HMMs that process symbol sequences instead of vector sequences. The transition weights a_{ij} represent the probabilities for selecting state s_j to emit the next feature vector if the previous feature vector has been emitted in state s_i. The entry vector e_i specifies the probability of emitting the first vector $\mathbf{x}_1$ of the feature vector sequence X in state s_i. Similarly, the exit vector e'_i specifies the probability for emitting the last feature vector $\mathbf{x}_T$ in state s_i. Transition probabilities, entry and exit vectors follow stochastic constraints:

$$a_{ij}, e_i, e'_i \geq 0, \quad \sum_j a_{ij} = \sum_i e_i = \sum_i e'_i = 1 \tag{2.3}$$

Because the sequence of acoustic feature vectors represents a temporal pattern, the HMMs used for speech processing usually only include transitions that are moving forward in time ($a_{ij} = 0$ for all $j < i$). A commonly used transition configuration is called "Bakis" model, which is depicted in figure 2.1. It connects every state with the following two states, allowing a self transition, a transition to the next state and skipping one state. The self transitions a_{ii} are of particular importance, because they allow arbitrary long subsequences of feature vectors to be associated with the same state. This eliminates the dependency between state index and feature vector index and considers all possible alignments between states and feature vectors.

Besides generating the multitude of possible state to feature vector alignments, the HMM is able to assign a probability to each of them. The generation of alignment hypotheses and their probabilities is done in a recursive manner: If there is an alignment hypothesis $(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \sigma_1, \ldots, \sigma_{t-1} = s_i)$ at state s_i and there is a possible transition to state s_j, the hypothesis is propagated further by multiplying its probability with the transition weight a_{ij} and the probability density value $p(\mathbf{x}_t|s_j)$ of the emission[2] of the current feature vector $\mathbf{x}_t$ in the destination state s_j:

$$\begin{aligned} p\left(\mathbf{x}_1, \ldots, \mathbf{x}_t, \sigma_1, \ldots, \sigma_{t-1} = s_i, \sigma_t = s_j\right) = \\ a_{ij}\, p\left(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \sigma_1, \ldots, \sigma_{t-1} = s_i\right)\; p\left(\mathbf{x}_t|s_j\right) \end{aligned} \tag{2.4}$$

[2]Strictly speaking, a probability density has to be integrated over a specific volume in the feature vector space to get a proper probability. For the sake of simplicity this is neglected, assuming a constant integration volume around the feature vector $\mathbf{x}_t$ with equal probability density $p(\mathbf{x}_t|s_j)$.

2.3 HMM parameter estimation

The most common way to approximate the emission probability densities $p(\mathbf{x}_t|s_i)$ is by means of Gaussian mixture densities:

$$p\left(\mathbf{x}_t|s_i\right) = \sum_m c_{m,s_i} N\left(\mathbf{x}_t, \boldsymbol{\mu}_{m,s_i}, \boldsymbol{\Sigma}_{m,s_i}\right) \tag{2.5}$$

$N(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the multidimensional Gaussian probability density with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

$$N\left(\mathbf{x}_t, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \; e^{-\frac{1}{2}(\mathbf{x}_t-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}_t-\boldsymbol{\mu})} \tag{2.6}$$

The problem of obtaining the parameters for each HMM of the recognition system is solved by estimating the parameters from a labeled speech corpus, which contains the orthographic transcriptions of all recorded utterances. If the corpus comprises a lot of utterances of a lot of speakers, it's possible to estimate model parameters that allow speaker independent speech recognition. The parameters are estimated by an iterative training procedure relying on the “Baum-Welch algorithm”, which allows to adjust the HMM parameters in a way that the acoustic model probability $P(X|W)$ is locally optimized for all training utterances.

The training of the HMM parameters is a demanding task, involving complex algorithms, which are difficult to develop. However, today there are efficient implementations of the training algorithms freely available, e.g. [YEK+02].

2.4 Isolated word recognition

If the speech recognition task is restricted to isolated words, the acoustic model probability $P\left(X|w\right)$ for a single word w can be computed by summing up the probabilities of all possible alignment hypotheses:

$$P\left(X|w\right) = P_{\mathrm{HMM}}\left(X|\lambda_w\right) = \sum_{\forall(\mathbf{x}_1,\ldots\mathbf{x}_T,\sigma_1,\ldots,\sigma_T)} p\left(\mathbf{x}_1,\ldots\mathbf{x}_T,\sigma_1,\ldots,\sigma_T\right) \tag{2.7}$$

The probability of production $P_{\mathrm{HMM}}(X|\lambda_w)$ of the word w by the corresponding HMM can be expressed recursively by the probability for aligning state s_j and feature vector $\mathbf{x}_t$, which is $p(\mathbf{x}_1,\ldots,\mathbf{x}_t,\sigma_t = s_j)$. Applying equation 2.4 and summing all “incoming” probabilities at state s_j yields:

$$p\left(\mathbf{x}_1,\ldots,\mathbf{x}_t,\sigma_t = s_j\right) = \left[\sum_i a_{ij} p\left(\mathbf{x}_1,\ldots\mathbf{x}_{t-1},\sigma_{t-1} = s_i\right)\right] p\left(\mathbf{x}_t|s_j\right) \tag{2.8}$$

which is also called forward probability, because the recursion is driven forwardly in time, from feature vector index $t-1$ to t. The initialization of the recursion is expressed by the aid of the entry vector e_i:

$$p\left(\mathbf{x}_1,\sigma_1 = s_i\right) = e_i p\left(\mathbf{x}_1|s_i\right) \tag{2.9}$$

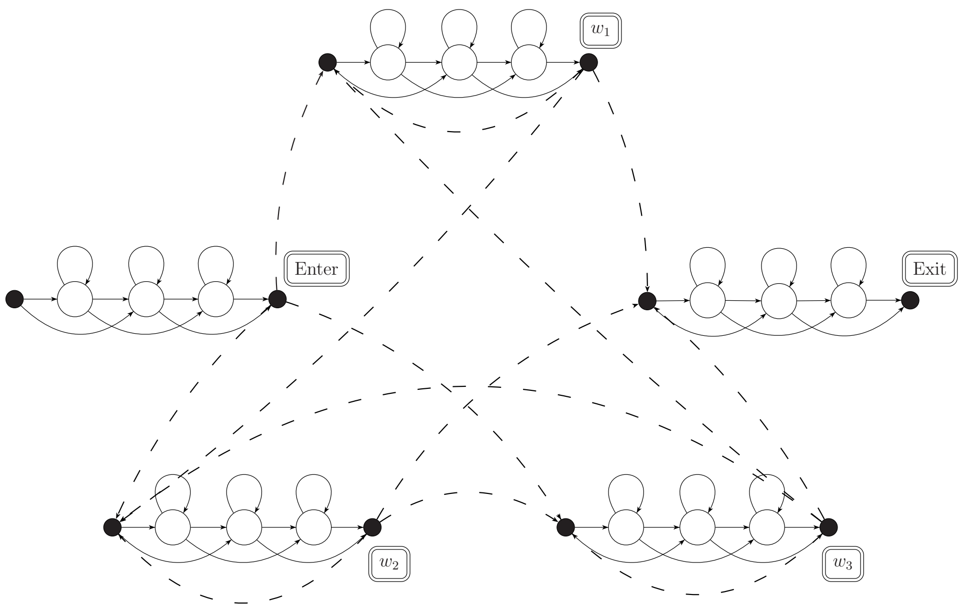

Figure 2.2: Example of integrated search network for continuous speech recognition. “Enter” and “Exit” refer to a silence model at the start and the end of the utterance.

The production probability $P_{\text{HMM}}\left(X|\lambda_w\right)$ is the sum of all $p(\mathbf{x}_1,\ldots,\mathbf{x}_T,\sigma_T=s_i)$ weighted with the exit vector e'_i:

$$P_{\text{HMM}}\left(X|\lambda_w\right)=\sum_i e'_i p\left(\mathbf{x}_1,\ldots\mathbf{x}_T,\sigma_T=s_i\right) \tag{2.10}$$

Because nearly every feature vector can be emitted in nearly every state, the actual alignment between states and feature vectors remains hidden, which is the reason for the term “hidden” Markov model.

Thus, in the case of isolated word recognition, the theoretical approach

$$w^*=\arg\max_w P\left(w\right)P\left(X|w\right)$$

for finding the most probable word w^* can literally be applied, which means to calculate the product of prior probability $P(w)$ and production probability $P_{\text{HMM}}(X|\lambda_w)$ for every word w and then selecting the word w^* with maximum joint probability.

2.5 Continuous speech recognition

For continuous speech recognition, the space of possible solutions cannot be enumerated, because the language model may allow arbitrary long word sequences. Therefore, the brute-force approach of assembling an HMM for every possible recognition outcome and naïvely applying equation 2.2 to find the word sequence with maximum joint probability becomes impracticable.

The solution for this problem is the consideration of the most probable alignment hypothesis between HMM state and feature vector sequence. Although the corresponding word sequence now is only an approximation for the theoretical solution given by equation 2.2, this approach has great importance in practice, because it leads to an efficient decoding algorithm, which operates on a single integrated search network combining language model and Hidden Markov Models.

2.5.1 Language model

The language model specifies a probability distribution over sequences of words. The most frequently used language model for continuous speech recognition is the n-gram language model (see [Jel76]). It assigns a probability for each word in the specific context of the preceding $n-1$ words. The probability for a word sequence W is the product of the n-gram probabilities for each word w_i:

$$P_{n-\text{gram}}(W) = \prod_i P\left(w_i | w_{i-n+1}^{i-1}\right) \tag{2.11}$$

The context w_{i-n+1}^{i-1} of the preceding $n-1$ words is also called history. The estimation of the n-gram probabilities requires a large text corpus from the application domain. The problem of data sparsity, which arises in this context, is handled by smoothing-techniques that allow to determine n-gram probabilities for histories that occur too rarely, or not at all in the training corpus. A comprehensive study of various smoothing-techniques can be found in [CG98].

2.5.2 Integrated search network

In order to build up the integrated search network, the language model itself has to be represented by a weighted transition network, which is possible for a wide range of commonly used language model types. Possible word sequences are represented by paths through this network, where every network node corresponds to a specific word in a specific context. The language model probability $P(W)$ for a particular word sequence W is calculated by multiplying the weights of the edges which are traversed by the corresponding path. Assuming unique HMM parameters for a particular word independent of its context in the language model network, the integrated search network is generated by replacing the word nodes by instances of the corresponding Hidden Markov Models, which is depicted in figure 2.2. For model interconnections every HMM instance in the integrated search network has an entry and exit node, which are also called “link” nodes.[3] These nodes interconnect HMM instances with a minimum number of network edges and don’t process any feature vectors like the emitting states s_i. An entry node forwards incoming alignment hypotheses to possible emitting states via the entry vector e_i, whereas an exit node receives alignment hypotheses leaving the model via the exit vector e_i'. The edges between exit and entry nodes of different HMM instances carry the weights of the corresponding edges in the language model network.

[3]There may be also link nodes in the language model network, which are also called “null” nodes and don’t represent a particular word. They remain in the integrated search network after the HMM replacement and are handled in the same way as entry and exit link nodes.

2.5.3 Time-synchronous Viterbi decoding

The decoding algorithm that operates on the integrated search network is known by the names "Viterbi algorithm" or "dynamic programming". It is based on a principle called "recombination", which means to propagate only the most probable alignment hypothesis at each considered node of the integrated search network. Concerning HMM states, the recombination principle can be expressed by the probability of the alignment hypothesis $p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \sigma_1, \ldots, \sigma_t = s_j)$, which survives in state s_j and emits the feature vector x_t:

$$\begin{aligned} & p\left(\mathbf{x}_1, \ldots, \mathbf{x}_t, \sigma_1, \ldots, \sigma_t = s_j\right) = \\ & \qquad \max_i \left[a_{ij}\, p\left(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \sigma_1, \ldots, \sigma_{t-1} = s_i\right)\right] p\left(\mathbf{x}_t | s_j\right) \end{aligned} \tag{2.12}$$

This implies the time-synchronous, or breadth-first organization of the Viterbi decoding algorithm, because all recombining alignment hypotheses must have processed the same number of feature vectors to have a fair comparison for choosing the best one.[4] The principle of recombination holds as well for the link nodes of the integrated search network, like entry and exit nodes. Because these nodes don't process any feature vectors, the difference to HMM states is that alignment hypotheses surviving in these nodes have to be propagated further in the same time step.

A well known way to organize time-synchronous Viterbi decoding on the integrated search network is called "token passing" (see [YRT89]), which in this context means the propagation of alignment hypotheses. A token represents an alignment hypothesis residing at a specific node or HMM state in the integrated search network. The propagation of a particular token means to send a copy of the token to each reachable node or HMM state and then to delete the original token.

In order to initialize the token passing, an initial token with probability one is put into the entry node of the integrated search network. Then, the token passing is carried out by the execution of the following steps for every feature vector $\mathbf{x}_t$:

- As long as there are link nodes which have a token attached to them, each of these tokens has to be propagated to the corresponding adjacent nodes or HMM states, multiplying its probability with the weight of the traversed edge, which is a language model weight or an entry vector weight e_i. Recombination takes place if the destination node or state already contains a token.

- Now there are only tokens residing at HMM states. The probability of each token is multiplied with the emission probability density $p(\mathbf{x}_t | s_i)$ of the corresponding HMM state.

- Each token at an HMM state is propagated to the adjacent HMM states or exit nodes, multiplying its probability with the corresponding transition weight a_{ij} or exit vector weight e'_i. Recombination again takes place if the destination node or state already contains a token. However, the difference to the propagation of tokens residing at link nodes is that incoming and outgoing tokens have to be handled separately, to prevent the asynchronous propagation of tokens.

[4]There are other decoding algorithms, like for example the A^* algorithm, which operates with a depth-first strategy. This thesis only considers the time-synchronous Viterbi decoding principle; for alternative approaches see [Ort98, Wil00].

An important extension of the decoding algorithm is the pruning of improbable alignment hypotheses. The pruning of hypotheses is effected during each time step of the decoding algorithm, where it discards all hypotheses having a probability score that exceeds the one of the currently best hypothesis by a specific value, which either can be a simple constant or may be adjusted dynamically.

2.5.4 Backtracking

Besides the propagation and recombination of tokens, it's important to keep book of the individual path which each token takes through the integrated search network. This allows to find out the recognition result, which is the sequence of words visited by the best alignment hypothesis. The best alignment hypothesis corresponds to the token that ultimately survives at the exit node of the integrated search network after processing the last[5] feature vector of the recorded user utterance. Because one is interested in the sequence of visited words, and not in the sequence of visited HMM states, the recording of traversed words only takes place at the exit nodes of the corresponding HMM instances. Usually, besides the traversed word, the current feature vector index is saved as well, which allows to determine the duration of every recognized word. The recording of words and feature vector indices is commonly organized by adding an entry in front of a linked list which is attached to each propagated token. This happens before the tokens are propagated to adjacent nodes in the first step of the token passing procedure. The recognized word sequence and the corresponding temporal segmentation is discovered by "backtracking", which simply means to reverse the linked list of words that belongs to the token which finally made its way to the exit node of the integrated search network.

2.5.5 Representation of probabilities

In practice, probabilities are represented on logarithmic scale, where they are called "scores". This is done for a better numerical representation of the small values resulting from the emission probability densities $p(\mathbf{x}_t|s_i)$. In order to control the influence of the language model, the language model weights are multiplied with a constant factor before they are added to the current score of a specific token. This language model factor usually is adjusted empirically by the evaluation of the speech recognition system on test utterances (held-out data, which is not used for the final evaluation).

2.6 Sub-word modeling

Modeling each word by a specific HMM with its own parameter set is only practicable in the case of a small recognition vocabulary, like for example digits or key words. For vocabularies comprising thousands of words, this modeling approach would result into a huge amount parameters, which cannot be estimated with sufficient statistical significance on the limited amount of training data.

[5]In the case of online speech recognition, the end of the utterance usually is detected by a speech activity detector.

This problem is solved by composing words by phonetic subunits. Because there are much less phonetic subunits than words, the HMM parameters can be estimated in a robust way. Further, the subunits can be composed to new words, which are not part of the training corpus. This allows a real separation of language model and acoustic model: The acoustic model training can be performed independently of the later application domain on utterances with arbitrary "phonetically rich" content.

2.6.1 Phonemes as sub-word units

Usually, words are composed of "phonemes", which are the smallest possible phonetic subunits[6]. The set of possible phonemes for a specific language has been defined and standardized by phoneticians. Most European languages comprise not more than about forty or fifty different phonemes. Possible pronunciations are usually passed to the speech recognition system in form of pronunciation lexica, which store possible sequences of phonemes for every registered word.

Creating pronunciation lexica for large recognition vocabularies is a tedious task, which is difficult to automate. Although there is an inherent correlation between orthography and pronunciation, there are also a lot of exceptions, which are hard to find out automatically.

2.6.2 Context-dependent phoneme models

Because the acoustic properties of phonemes depend a lot on the context of articulation, an independent model for all possible contexts is only a crude approximation. A common method for a better representation of the effect of coarticulation is the use of context-dependent phoneme models. In the case of "triphone" models for example, phonemes of the same class are modeled with a particular HMM parameter set for each possible combination of left and right context phonemes that appears in the pronunciation lexicon. The problem of data sparsity, which inhibits the estimation of distinct parameters for every possible triphone combination, can be solved by a method called "decision tree clustering" (see [Beu99]). This method is based on the strategy of "parameter tying", which means to share parameters in different models. A phonetic decision tree, which is built up on the training data, allows to find out, which HMM parameters can be shared between different contexts of the same phoneme. Furthermore, it permits to assemble HMM parameters for triphone combinations that didn't occurred in the training data at all, which preserves the independence of training and recognition vocabulary.

The extension of context-independent phonemes to triphones is often restricted to word internal phonemes. In this case, phonemes at word-boundaries are only extended to biphones, respectively left as they are if the word only consists of a single phoneme. This preserves the assumption that there is only a single acoustic-phonetic model for each word, independently of possible neighboring words which are predicted by the language model. So-called "cross-word triphones" allow the correct context extension even at word boundaries, which however makes the construction of the integrated search network much more complicated.

[6]The term "phoneme" denotes the class of a specific speech sound; the speech sound itself is also called "phone". Therefore, a phone is the realization of a specific phoneme.

Chapter 3

Modeling approach for one-stage speech interpretation

This chapter introduces the knowledge representation framework for semantic interpretation which is used in the context of this thesis. It follows the standard method of semantic grammars, which directly capture application-specific semantics, without explicitly separating syntax and semantics. The elected representation of grammatical constraints in the form of a weighted transition network hierarchy has the primary purpose to facilitate the intended tight coupling of speech recognition and parsing (one-stage speech interpretation). Moreover, this representation allows a straightforward incorporation of stochastic weights, which can be estimated from collected application-specific data.

The chapter begins with a brief introduction to the theory of "formal languages" (see [HU79]), which provides the basis for the following sections that discuss the modeling framework used for semantic interpretation and point out the differences with respect to other approaches that propose more sophisticated formalisms. The presentation of the modeling framework will be given from the perspective of its application and does not provide details on the creation process of the semantic-syntactic model, which is one of the main subjects of Thomae's thesis (see [Tho06]).

3.1 Representation and processing of formal languages

A formal language comprises a set of symbol sequences which all follow specific constraints. The simplest way to define a formal language is to write down the set of valid symbol sequences. However, valid symbol sequences may have arbitrary length. Therefore, it is necessary to have a compact representation of formal language constraints. This compact representation can be provided by formal grammars or abstract machines. The following sections explain both representation methods and their strong relation.

3.1.1 Context-free and regular grammars

Grammars are made of production rules that define possible transformations of symbol sequences by the replacement of non-terminal symbols by sequences of non-terminal and terminal symbols. The repeated application of production rules generates sequences of terminal symbols which are declared to be part of the corresponding formal language. The application of production rules starts with an initial non-terminal symbol and produces a valid sequence of terminal symbols each time when after a specific sequence of rule expansions all non-terminal symbols have been replaced by terminal ones. This leads to the problem of enumerating valid symbol sequences that belong to the formal language in a specific order. On the other hand, one can ask if a given symbol sequence belongs to the formal language. The operation which systematically solves this problem is called "parsing".

Formally, a grammar $G = (N, \Sigma, S, P)$ is specified by the alphabet N of non-terminal symbols, the alphabet Σ of terminal symbols, the initial non-terminal symbol S and a set of production rules P. Different constraints on production rules yield different types of grammars. The resulting classification scheme that ranks grammars and corresponding formal languages due to their expressive power has been introduced by the famous linguist Chomsky and therefore is called Chomsky hierarchy (see table 3.1).

The most general type of grammar, which is on the top of the Chomsky hierarchy, are "unrestricted" grammars. They allow production rules of the form $\alpha \rightarrow \beta$, where α and β denote arbitrary sequences of non-terminal and terminal symbols with the only constraint that α contains at least one non-terminal symbol.

When going down the Chomsky hierarchy, the next lower level always represents a specialization of the grammar type on the corresponding higher level. The first specialization are "context-sensitive" grammars. They have production rules of the form $\alpha A \beta \rightarrow \alpha\gamma\beta$, which means that a non-terminal symbol A which occurs in the context of the symbol sequences α and β can be replaced by the symbol sequence γ, with the constraint[1] that γ cannot be the empty symbol ϵ.

Theoretically, both unrestricted and context-sensitive grammars have the power of describing complex languages, like natural languages. Paradoxically, it is just their own superior expressive power which makes the actual use of such grammars difficult. The consistent specification, as well as the algorithmic processing of the huge set of necessary rules becomes intractable in practice.

Therefore, the grammar types with less expressive power, which are situated at the bottom of the Chomsky hierarchy have much greater impact on practical applications than unrestricted or context-sensitive grammars. These grammars are called "context-free" and "regular" grammars. A context-free grammar has production rules of the form $A \rightarrow \gamma$, which means that a non-terminal symbol A can be replaced by a sequence of non-terminal and terminal symbols γ, independently of the context in which occurs A.

Regular grammars have the lowest expressive power in the Chomsky hierarchy and include context-free grammars with the constraint that their production rules can be transformed by the introduction of intermediate non-terminal symbols into a set of rules where all rules have either the form $A \rightarrow aB$ and $A \rightarrow a$ or the form $A \rightarrow Ba$ and $A \rightarrow a$. The distinction between context-free and regular grammars becomes more obvious when looking at their equivalent automaton representations,

[1]Without this constraint, it can be shown that context-sensitive rules would have the same expressive power as unrestricted production rules.

grammar type	equivalent machine	production rules	example
unrestricted	Turing machine	$\alpha \rightarrow \beta$ $\alpha, \beta \in (N \cup \Sigma)^*$	(see footnote)
context-sensitive	linear-bounded automaton	$\alpha A \beta \rightarrow \alpha \gamma \beta$ $\alpha, \beta, \gamma \in (N \cup \Sigma)^*$ $A \in N, \gamma \neq \epsilon$	$\{a^n b^n c^n \| n > 0\}$
context-free	push-down automaton	$A \rightarrow \gamma$ $A \in N, \gamma \in (N \cup \Sigma)^*$	$\{a^n b^n \| n > 0\}$
regular	finite-state automaton	$A \rightarrow aB$ or $A \rightarrow Ba$ $A \rightarrow a$ $A, B \in N, a \in \Sigma$	$\{a^m b^n \| m, n > 0\}$

Table 3.1: Chomsky's hierarchy of formal grammars and their equivalent abstract machines.

which is done in the next section. Table 3.1 shows the Chomsky hierarchy including typical examples for formal languages that require[2] the corresponding grammar type to be specified.

3.1.2 Finite-state automata and regular expressions

The other way to define formal languages is by means of abstract machines. The simplest one, which is called "finite-state" automaton comprises a set of states and a set of transitions between states. Transitions are associated with symbols from the alphabet of the formal language. Like grammars, abstract machines can be regarded from the point of view that they can generate valid symbol sequences of the corresponding formal language. In the case of the finite-state automaton, valid symbol sequences correspond to paths through the automaton, beginning from a specific initial state and ending at specific final states.

On the other hand, asking the question whether a given symbol sequence is part of the formal language, leads to the problem to decide if the corresponding abstract machine "accepts" the given symbol sequence. For a finite-state automaton this means to find out whether there is a path from the initial state to a final state which matches the given symbol sequence.

Formally, a finite-state automaton $A = (\Sigma, Q, E, i, F)$ is defined by

- the alphabet Σ of the formal language,
- the set of states Q,
- the set of transitions $E \subset Q \times (\Sigma \cup \epsilon) \times Q$, which is a subset of all possible combinations of source and destination states and symbols including the empty symbol ϵ,

[2]From literature it doesn't get quite clear if there is a simple example for a formal language that definitely requires an unrestricted grammar to be specified.

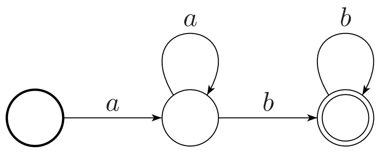

Figure 3.1: Finite-state automaton accepting the language $\{a^m b^n | m, n > 0\}$

- the initial state $i \in Q$,
- and the set of final states $F \subset Q$.

The common way to represent finite-state automata graphically, is shown in figure 3.1. Transitions are depicted by arcs that connect source and destination states and carry a specific symbol. The initial state is indicated by bold print and the final states by concentric circles.

As indicated by the Chomsky hierarchy in table 3.1, it can be shown that finite-state automata have the same expressive power like regular grammars. Therefore, the subclass of regular languages is simply the class of formal languages which can be represented by finite-state automata.

All other grammar types have equivalent abstract machines, too. For instance, the example language $\{a^n b^n | n > 0\}$ does not belong to the subclass of regular languages and therefore cannot be represented by a finite-state automaton. The abstract machine representation for this context-free language requires the concept of a push-down automaton. This is a finite-state automaton which is extended by a stack that can store symbol sequences from an auxiliary alphabet of stack symbols. In addition to the symbol of the language alphabet, transitions between states are associated with the symbol which is currently on the top of the stack and with a specific manipulation of the stack itself. Higher-order abstract machines further relax the constraints on the storage device. For instance, Turing machines allow the manipulation of an infinite tape of symbols.

As reflected by their complicated description, abstract machines don't provide a practical way for manually specifying language constraints. For humans, the specification of automata is usually more difficult to manipulate than their equivalent grammatical representation. Just like higher-order grammars, higher-order abstract machines are mainly of theoretical interest. Apart from the problem of specifying such machines, elementary problems, like deciding if a symbol sequence is accepted or not, become intractable or even unsolvable.

In contrast to higher-order abstract machines, finite-state automata are of great importance in practice. Their inherent principle of connectivity captures many practical problems and leads straightforwardly to algorithms that can efficiently solve these problems. For example, the methods used for automatic speech recognition, which have been presented in the last chapter, are closely related to finite-state automata.

As stated before, context-free production rules do not obviously reflect the constraint of regular languages. This is what can be achieved by "regular expressions". Just like the attribute "regular" implies, this notation exactly captures regular lan-

operation	common notation	explication
concatenation	ab	a followed by b
choice	$a\|b$	a or b
option	$[a]$	a or nothing
repetition	$a+$	one or several a's
optional repetition	$a*$	the same as $[a+]$
bracketing	$a(b\|a)$	scope and precedence

Table 3.2: Operations used in regular expressions.

guages and thus provides an equivalent representation of finite-state automata. Regular expressions are based on a calculus defined on symbols. Table 3.2 shows the set of operations which are commonly used in regular expressions. Regular expressions allow the specification of regular language constraints in a human readable form. They also facilitate the manual specification of context-free grammars. The resulting specification format, which is known by the name "extended Backus-Naur form" (EBNF), compactly represents possible productions of a particular non-terminal symbol by a corresponding regular expression.

An important extension to finite-state automata is the introduction of output symbols, which allows to translate symbol sequences over a source alphabet into symbol sequences over a destination alphabet. The production of output symbols can either be associated to states or to transitions, yielding two different finite-state automaton representations, which are called "Moore", respectively "Mealy" representation.[3] Both representations are equivalent and are convertible into each other (see [HU79]). Because of their ability to translate symbol sequences from one alphabet into another, this kind of automata are also called "transducers". They are of particular interest in the context of this thesis, because they provide the basis for an efficient integration of speech recognition and interpretation (see section 4.2).

3.1.3 Parsing algorithms

As mentioned earlier, parsing solves the problem to decide if a specific symbol sequence belongs to the formal language which is given by a formal grammar. Parsing algorithms systematically try to reveal the sequence of production rules that lead from the initial non-terminal symbol to the given sequence of terminal symbols. If such a derivation can be determined, the symbol sequence is part of the formal language. In practice, the derivation itself is of major interest. The "parse tree" reflects the grammatical structure of the symbol sequence, which is exploited for the actual purpose of parsing, namely to translate the symbol sequence into some other representation.

There are many parsing algorithms that differ in various aspects. For instance, the strategy to reveal the parse tree either bottom-up, starting from the given symbol sequence or top-down, starting from the initial non-terminal symbol, distinguishes

[3]The term "Moore"-automaton is also used for finite-state automata with a single alphabet and symbols that are attached to states instead of transitions.

the classes of LR and LL parsing algorithms. LR-parsers are often involved in the construction of compilers that translate source code into executable machine code. LL and LR parsers can only process particular subclasses of context-free languages and cannot output multiple parse trees in case of ambiguous grammars. This restrictions don't have much relevance when processing purely formal input, like programming languages, but get important for natural language processing. For this purpose serve chart parsing techniques, like the Early algorithm (see [Ear70]), which don't have these restrictions. However, instead of linear complexity, chart parsers have cubic complexity with respect to the length of the input symbol sequence.

An important extension is offered by robust parsing algorithms which have the ability to process ungrammatical input. They can find a partial derivation by skipping terminal symbols that do not match the given formal grammar (see [Abn96]).

3.1.4 Recursive transition networks

Recursive transition networks are the logical consequence of the compact representation of context-free production rules by regular expressions. Having in mind that regular expressions are equivalent with finite-state automata, a context-free grammar can intuitively be interpreted as a hierarchy of finite-state automata. Each transition (or state, if preferring the Moore automaton representation) which is labeled with a non-terminal symbol refers to the corresponding sub automaton.

Although being illustrative, the idea of a hierarchy of transition networks is not entirely correct, because production rules may recursively refer to each other. Therefore, the term "recursive transition network", which has been introduced by Woods (see [Woo70]), is preferred in literature. This term, however, does not imply that it actually refers to a collection of transition networks.

In this thesis, the recursive transition network representation is of particular interest, because it provides a straightforward way to impose context-free grammar constraints on the standard decoding method used for speech recognition. This avoids the complex modification of a symbolic parsing algorithm, like the Early-algorithm, which is necessary to make the parser work on the non-symbolic input given by the recorded speech signal (see section 3.3.1).

3.1.5 Stochastic weights

The introduction of stochastic weights is an important extension to the framework of formal language description. The most obvious application of stochastic weights in this context is the solution of the problem of disambiguation: If a grammar allows more than a single derivation for a specific input, possible solutions may be ranked due to their probability, which allows to select the most probable derivation.

When dealing with uncertain input, the purpose of stochastic weights goes further: in the case of speech recognition, a prior distribution over possible word sequences is combined with the statistical models used for the recognition of acoustic-phonetic patterns. As explained in chapter 2, this method allows to determine the solution that best meats the given constraints over all modeling levels.

Of course, the use of stochastic weights relies on collected example data which allows their estimation. This estimation assumes a specific model of syntactic constraints. A simple model that is frequently used in speech recognition, is the n-gram model, which provides a probability for a word given the $(n-1)$ predecessor words (see section 2.5.1). The stochastic weights can be determined by quite simple counting and smoothing techniques and the model can be represented by a weighted finite-state automaton.

Also the definition of context-free grammars can be extended to include stochastic weights. This is done by assigning a probability to every possible production of a specific non-terminal symbol. Following the stochastic constraint, all probabilities for a particular non-terminal must sum to one. The estimation of the grammar weights by means of simple counting requires a collection of parse trees for all example phrases.

To build up such a collection of parse trees may afford a considerable amount of manual work, which would be nice to avoid. There is actually a method which allows the estimation of grammar weights without the explicit need for annotated parse trees. It is the iterative "inside-outside" algorithm which is based on the same "expectation maximization" principle (EM) like the Baum-Welch training algorithm that is used for the training of Hidden Markov Models (see [LY90]). Theoretically, this estimation algorithm even includes the possibility for the structural inference of grammar rules. Without imposing any further constraints on the structure of the grammar, the totally data-driven algorithm however fails to infer rules which may serve for the purpose of semantic interpretation (see end of section 3.3.3).

3.2 Generalizing vs. application-specific semantics

After the short overview of formal language theory, this section presents its application to the task of semantic interpretation, which is in the focus of this thesis. The section will point out the difficulties arising from linguistically motivated approaches to semantic interpretation which justify the application-specific approach of semantic grammars which is used in this thesis.

3.2.1 Feature grammars and first order logic

Motivated by the underlying theory of linguistics, the the natural language processing community regards semantic interpretation as a formalism that relies on the syntactic structure of a sentence, which can be described by a formal grammar. Such a grammar contains non-terminal symbols that represent linguistic phrase units, like noun or verbal phrases, and word categories, like nouns or adjectives. However, formal grammars do not serve well for specifying the syntax of natural language, which usually affects the morphology of the underlying words. This is not necessarily a consequence of the limited generative power of the used formal grammar type. Actually, linguists argue about the question whether the class of context-free grammars suffice to represent the constraints of natural language. Regardless of this question, the actual problem in practice is the systematical specification of the large set of necessary rules. To address this problem, the original definition of production rules has been augmented to allow the specification of particular "features" that accompany non-terminal and terminal symbols. Such features may represent syntactic properties like person and number (e.g. "third person singular") and are used to restrict valid grammatical derivations by imposing the condition of identical feature values for all involved constituents. The use of features allows a compact representation of syntactically valid phrases that result from the set of all possible feature value configurations. Beyond the specification of syntactic constraints, the feature paradigm can also be used for semantic interpretation. The evaluation of semantic features during parsing allows the generation of logical forms that represent semantics by means of first-order logic. First order logic is a quite general formalism to represent semantic knowledge and which allows to specify queries that can automatically be answered by an inference algorithm.

Actually, the last paragraph does not refer to a specific approach. It is intended as an outline of a whole class of approaches which follow the tradition of natural language processing. These approaches lead deeply into the field of artificial intelligence and relate closely to expert systems based on logic programming. For a detailed introduction into this subject, see [All95].

Anyway, the common objective of all linguistically motivated approaches is to achieve a high degree of generality with respect to the class of representable syntactic constraints and semantic content. However, the specification and the handling of the complex set of augmented grammar rules require highly specialized human experts. Unfortunately, up to now, it has not been achieved to provide front-ends for such generalizing descriptions of syntax and semantics, which could easily be used by non-experts to specialize the general knowledge representation for the realization of a particular application.

Another substantial problem arises when dealing with spoken language. User utterances in the context of practical applications may simply not be grammatical from the linguistic point of view, which puts the great effort on syntactic correctness into question.

3.2.2 Semantic grammars and slot-value pairs

The fact that linguistically motivated approaches to semantic interpretation are difficult to use in practice is reflected by the situation that such approaches are not part of the evolving industry standard for the specification of spoken dialog systems of the world wide web consortium W3C (interaction domain "Voice Browser" Activity, see [W3C04]).

The W3C standard favors the use of "semantic grammars", which are context-free grammars with a set of non-terminal symbols that directly capture application-specific semantics. Syntactical or morphological constraints are only considered as far as they directly affect this application-specific semantics. The proposed semantic representation of user utterances happens by collections of simple variables, which in the following are called slot-value pairs. Their creation is controlled by scripting commands which are embedded into the regular expressions that specify the right hand side of context-free production rules. While the parser applies a specific production rule, these scripting commands are executed in order to create new slots that are filled with an arbitrary combination of hard-coded information and the values of slots which have been extracted during the application of subordinate production rules. Scripting facilities, like arithmetical operators or string processing commands, provide a flexible way to extract information and to translate it into the

```
public $WC_Hour24 =
  ein { 1 } | zwei { 2 } | ... | dreiundzwanzig { 23 } ;
...
public $C_MinuteRel =
  viertel vor   { -15 }
| viertel nach  {  15 }
| drei viertel  { -15 }
| halb          { -30 }
...
public $C_Time =
  $WC_Hour24 { h = $WC_Hour24; m = 0 } Uhr
    [ $WC_Minute { m = $WC_Minute }]
| $WC_Hour24 { h = $WC_Hour24; m = 0 }
    $WC_Minute { m = $WC_Minute }
| $C_MinuteRel $WC_Hour12 { h = $WC_Hour12; m = $C_MinuteRel } ;
```

Figure 3.2: Snippet of a semantic grammar in ABNF format that describes the expression of the time of day in German language.

desired format that for example meets the requirements of later data base queries.

A severe problem in domain-independent approaches to semantic interpretation is ambiguity: words or phrase parts may have several meanings which depend on the pragmatic context of the corresponding utterance. In the limited domain of a specific dialog application, however, user utterances usually have a low degree of ambiguity. Therefore, ambiguous grammar rules can be avoided and the few cases of slot-value pairs with ambiguous interpretations are handled by the dialog system with initiating a dialog step that asks the user to resolve the ambiguous piece of information. Accidental rule ambiguities which may occur due to an imperfect design of the manually crafted interpretation grammar, can be resolved by the stochastic weighting of the grammar (see section 3.3.3).

Figure 3.2 shows a snippet of a semantic grammar in the standardized ABNF format (augmented Backus-Naur form, see [W3C04]) that provides a rudimentary representation of possible expressions of the time of day in German language. For example, when parsing the word sequence “drei Uhr zehn” (ten past three) the result in form of the slot-value pair collection is $\{h = 3, m = 10\}$. Inside a slot filling command the reference to a non-terminal symbol addresses the slots which have been created during the application of the corresponding subordinate production rule. The interpretation of “halb drei“ (half past two), which is $\{h = 3, m = -30\}$, is intended to give an example for the case that the interpretation formalism does not suffice to translate the information into the final format which may afford some kind of post-processing on the extracted collection of slot-value pairs.

Semantic grammars surely have many weak points, like the high dependency on the application domain or the inability to represent complex semantical relations. But they allow the rapid design of practical applications from scratch without the requirement that the application developer is an expert in computational linguistics.

3.3 One-stage speech interpretation

As already pointed out in the introduction, this thesis follows the standard approach to semantic-syntactic modeling via the use of semantic grammars, but extends the basic idea in two aspects: The first aspect is the choice of a representation which favors the tight coupling of speech recognition and parsing, which is called one-stage speech interpretation. The second aspect concerns the consideration of stochastic weights. This section introduces the utilized tight coupling approach and describes the selected representation of the semantic grammar and the corresponding stochastic weighting in form of a weighted transition network hierarchy (WTNH).

3.3.1 Tight coupling approach

As pointed out in the introduction, the sequential coupling of speech recognition and parsing involves severe problems, which can be avoided by a tight integration of parsing and acoustic-phonetic decoding. However, a parsing algorithm inherently works on symbolic input, which is not provided when having to process the recorded speech signal. Therefore, the parser has to be extended to consider acoustic-phonetic models, which are usually Hidden Markov Models (see section 2.2). Unfortunately, there is no straightforward solution to this problem, because the evaluation of the acoustic-phonetic models neither provides certainty about actual symbol identities, nor certainty about the temporal boundaries between symbols. Thus, the parser has to keep track of multiple hypotheses that correspond to different time alignments of possible input symbol sequences. Among these hypotheses the parser has to find out the one that best matches the recorded speech signal. During this search process, hypotheses have to be propagated for every extracted feature vector that corresponds to a speech signal segment of about ten milliseconds. Therefore, the actual challenge of tight coupling is to find a way to keep the bookkeeping of hypotheses computationally tractable.

About the question whether tight coupling is feasible or not, one can find some kind of controversy in technical literature. Since the early nineties, many tight coupling approaches have been published for various types of parsing techniques and different types of grammars or even more sophisticated semantic-syntactic models (e.g. [KTK89, Ney91, GZ92, JWT$^+$95, WFS97, Sta97]). Nevertheless, there have been doubts about the computational tractability of such approaches due to the cubic time complexity of the involved parsing algorithms with respect to the size of the input, which instead of the number of words is now determined by the much greater number of extracted feature vectors (see [HJM$^+$94, Rin95]). However, this argument may underestimate the positive effect of the pruning of improbable hypotheses, which is one of the great advantages of an integrated decoding algorithm.

In contrast to the just mentioned approaches, the tight coupling approach which is presented in this thesis, avoids the use of a parsing algorithm and takes advantage of the equivalence of a context-free grammar and a hierarchy of transition networks. The shift from grammar rules to transition networks avoids the complex integration of the acoustic-phonetic decoding into the architecture of a symbolic parser and allows to use the Viterbi decoding principle which is commonly applied in speech recognition (see section 2.5.3). This kind of strategy has first been suggested by

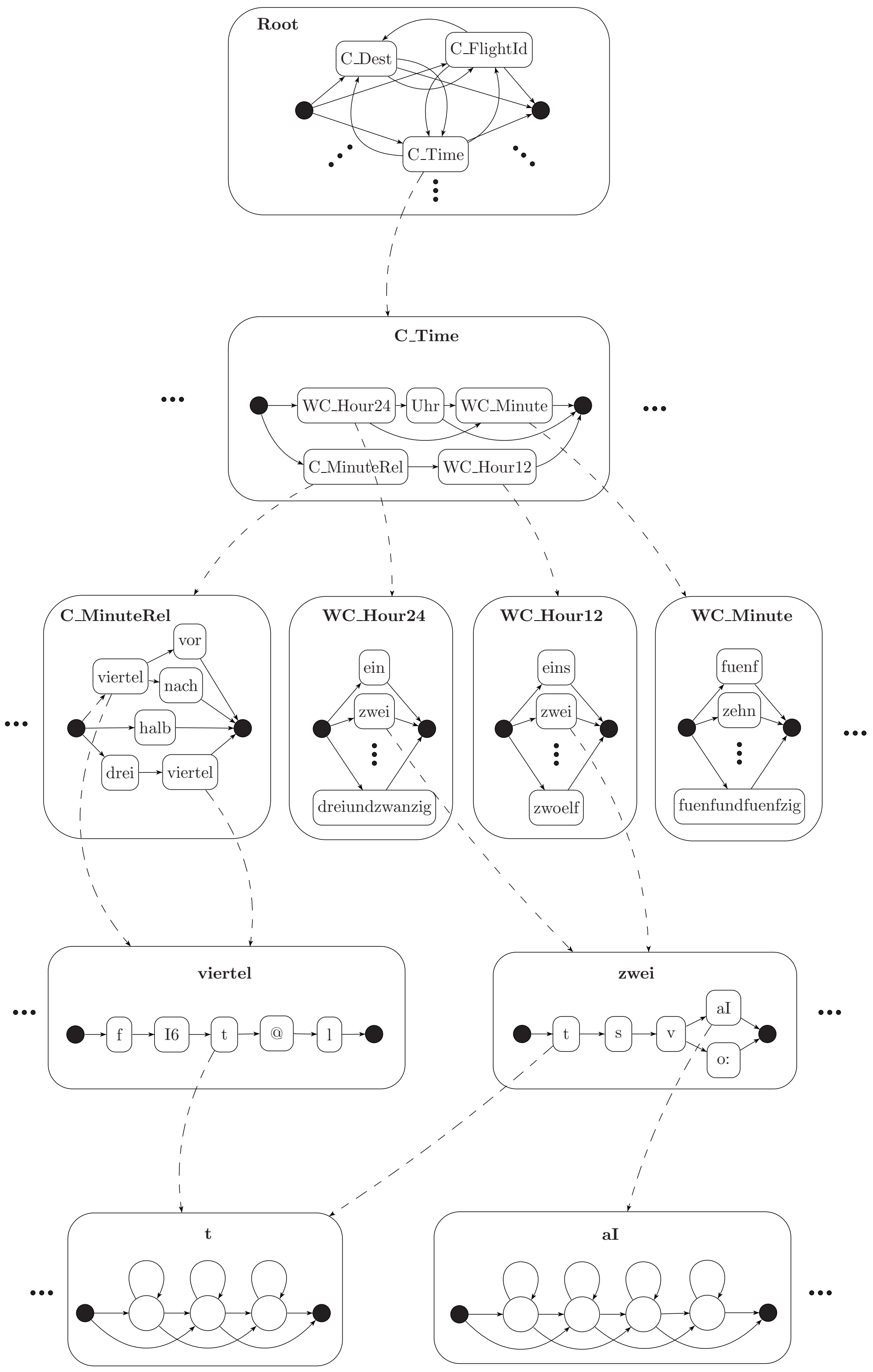

Figure 3.3: Snippet of exemplary WTNH representing all relevant knowledge sources on semantic-syntactic, lexical and acoustic-phonetic level.

Moore, Pereira and Murvit (see [MPM89], and also [Dup93]).

3.3.2 Weighted transition network hierarchy (WTNH)

The selected representation of semantic grammars in form of a hierarchy of weighted transition networks provides a uniform modeling framework which is also suited for the representation of the lexical and acoustic-phonetic knowledge sources. The pronunciation of specific words, as well as the Hidden Markov Models for the acoustic modeling of specific phonemes can be represented by weighted transition networks and thus fit into the network hierarchy.

Figure 3.3 shows a snippet of an exemplary WTNH that models user utterances for the airport information system, which is the application scenario used in this thesis (see section 6.2). The figure particularly includes those subnets that correspond to the semantic grammar part given in figure 3.2, which describes the time of day in German language. By convention every subnet has a single entry and a single exit node which do not refer to any subnet. The dashed edges connect subnet nodes (or parent networks) with their corresponding subnets. Regarding these edges, subnets can also be interpreted as nodes of a "super network" which represents the dependencies between parent and child networks.

The WTNH representation provides the basis for a one-stage decoding algorithm that tightly integrates all included knowledge sources and is able to determine the best-matching parse tree directly from the recorded speech signal. The final result in form of the slot-value pair collection can be extracted by applying the semantic grammar along the decoded parse tree. A detailed discussion of the actual realization of the one-stage decoding algorithm will be given in the next chapter. The remaining part of this chapter briefly describes the generation process of the "hierarchical language model" (HLM, for details see [Tho06]).

The HLM is the semantic-syntactic part of the WTNH which corresponds to the semantic grammar. Due to the equivalence of regular expressions and finite-state automata, the structure of each subnet of the hierarchical language model can automatically be constructed from the regular expression that specifies the right hand side of the corresponding grammar rule. The AT&T Lextools served for the conversion of regular expressions into finite-state automata (see [Spr03]); a detailed description of the conversion algorithm can be found in [HU79].

3.3.3 Creation of the hierarchical language model (HLM)

The training procedure for the HLM described in [Tho06] is effected on a set of user utterances collected during an experiment which simulates the spoken dialog application (Wizard-of-Oz experiment, see section 6.2). Beyond the orthographic transcription, the annotation of each user utterance has to include the parse tree which follows the semantic grammar that has been developed for the extraction of the slot-value pairs. Actually, the primary knowledge source in [Tho06] is not the semantic grammar, but the hierarchical annotation which is constructed manually with the aid of graphical annotation tools. In the context of this thesis, however, the collection of parse trees is created automatically by parsing the orthographic corpus transcription with the hand-crafted semantic grammar. The training pro-

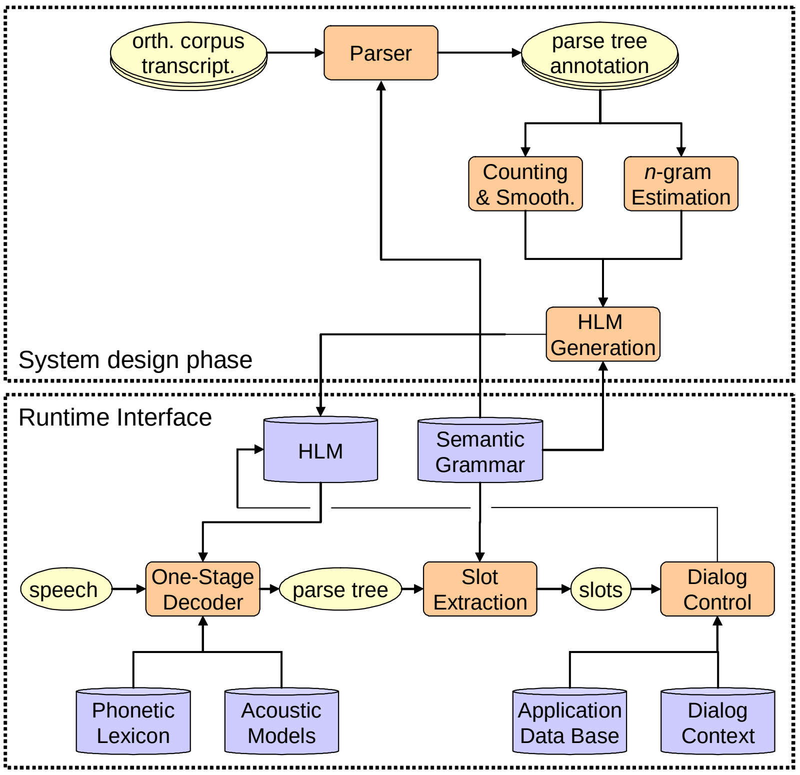

Figure 3.4: Generation and application of the hierarchical language model (HLM) in the context of a spoken dialog system.

cedure for the HLM is used to estimate the stochastic weighting for the transition network hierarchy which is automatically created from the structure of the semantic grammar. Figure 3.4 shows a block diagram that points out the generation and also the application of the HLM inside a spoken dialog system (see also [LTR^{+}05]).

The main assumption of the training procedure is the independence of the stochastic weighting of a subnet from the context of its use in possible parent networks. The hierarchical annotation in form of the parse trees provides the set of all appearing sequences of subnet nodes for every subnet that represents a non-terminal symbol of the semantic grammar. This assumption of independence allows to apply conventional language modeling techniques to estimate the distribution of weights in every subnet from the observed collections of subnet node sequences.

For each subnet, the grammar developer can choose from the following weighting techniques:

- **n-gram weight distribution.** For the case that a rule of the semantic grammar allows an arbitrary sequence from a set of specific non-terminal or terminal symbols, the corresponding subnet can be constructed from a n-gram model estimated over the set of observed subnet node sequences. An example for such a rule is the head rule of the semantic grammar, which usually allows an arbitrary sequence of main rules, to permit the user to pass a variable number of semantic slots in a single utterance. This is indicated by the topology of the top-level network in the example for the WTNH in figure 3.3.

Any n-gram model can automatically be converted into a finite-state network, for example using the SRILM toolkit [Sto02]. In this way the HLM allows the combination of n-gram and context-free grammar constraints. To cope with the problem of data sparsity, various smoothing techniques have been tested in [Tho06] with an HLM for the airport information application domain, however without observing great differences in the obtained performance improvements. For the experiments carried out in the context of this thesis, the smoothing was done with Katz's method yielding a so-called "back-off" n-gram model (see [Kat87]).

- **Counting and smoothing.** If a grammar rule translates into a network with several different path alternatives that correspond to wordings that are used differently often, these paths can be weighted according to their occurrence in the training corpus. The weight of a network edge is estimated from the number of traversals counted while walking through the parse trees of the training corpus. To take into account unused parts of the semantic grammar and to balance unreliable counts caused by data sparsity, the resulting weights are smoothed with the Good-Turing discounting method (see [Goo53]).

- **Uniform weight distribution.** There are also many grammar rules whose corresponding subnets should not be weighted from the training corpus, because all paths are equally probable and their distribution in the training corpus has no general significance. This mainly concerns subnets which represent information from the application data base. As indicated in figure 3.4, the content of such subnets may also be updated by the dialog management module during the runtime of the spoken dialog system. To achieve a uniform weight distribution in these subnets, the outgoing edges of every network node are weighted equally.

Another important issue are natural speech disfluencies, like pauses, non-speech sounds, filler words, cut-off words or self-corrections.[4] In order to prevent the misinterpretation of these disfluencies, they should be included in the HLM. For pauses and non-speech sounds there are explicit Hidden Markov Models, just like for the normal phonemes. Filler words, like hesitations ("uhm") are modeled by special lexical entries. Cutoff words or general out-of-vocabulary words can be represented by lexical models that predict possible phoneme sequences (see [TFLR05]).

Besides the creation of acoustic and lexical models for natural speech phenomena, there is also the problem to integrate them into the HLM. To release the grammar developer from guessing where inside the semantic grammar he has to include references to filler and out-of-vocabulary models, the location of such references is automatically determined from the corpus annotation. For this purpose natural speech phenomena have to be annotated during the orthographic transcription of the user utterances. The parser which is used to generate the parse tree annotation

[4]Self-corrections are handled by the n-gram top-level network, which allows the arbitrary repetition of main rules; the value of the corresponding semantic slot is simply overwritten in the order of occurrence.

of the orthographic corpus transcription is a robust parser[5] which is able to cope with partly ungrammatical utterances. It allows the occurrence of sequences of filler words and unmatched words between grammatical correct words and determines the parse tree that includes the minimum number of unmatched words. During the generation of the structure of the HLM from the semantic grammar, subnet nodes referring to filler or out-of-vocabulary (unmatched) words are automatically inserted at the locations which have been determined by the robust parser. This is actually done by merging each subnet which is constructed from a specific grammar rule with the subnet constructed from the collection of annotated child node sequences, which can be found for the corresponding non-terminal symbol in the corpus of parse trees.

The integration of an out-of-vocabulary model is optional. An alternative for avoiding the misinterpretation of ungrammatical utterance parts is the estimation of semantic confidences, which will be presented later on (see sections 5.3 and 6.3.2).

Admittedly, the suggested creation process of the HLM affords a considerable amount of manual work, which has to be done by the responsible grammar developer. This work includes the crafting of the semantic grammar from the abstract specification of the particular dialog application and its test and refinement on the application-specific corpus, which also has to be collected in advance. Furthermore, for the estimation of the stochastic weights of the corresponding HLM, the kind of weighting (n-gram, counting and smoothing, or uniform distribution) has to be selected manually for every subnet. On the other hand, this approach provides a strict control of the resulting model which prevents situations where the developer is not able to fix a particular failure situation, because he cannot influence the outcome of some sophisticated rule inference algorithm.

Nevertheless, approaches to semantic-syntactic modeling which afford less handcrafted work are definitely desirable. As already mentioned in section 3.1.5, there is a training algorithm for stochastic grammars that has the ability of structural inference (inside-outside algorithm, EM principle, see [LY90]). However, the unconstrained version of this algorithm, which is totally data-driven, is not able to determine a grammatical structure that can serve for the purpose of semantic interpretation. Thus, it is crucial for practical applications to put tight constraints on the degree of freedom of the assumed grammatical structure to achieve a systematical relation with the semantic content that finally has to be extracted. In [HY05] and [WDA05] one can find promising approaches for the application of an EM-based training algorithm that assumes a restricted context-free grammar structure. These approaches only require a specification of the dominance relationships of the set of necessary non-terminal symbols and a semantic annotation for every training utterance that serves as an additional constraint during the iterative training procedure. The semantic annotation of a specific corpus utterance only includes the hierarchy of the involved non-terminal symbols and does not demand a manual specification of the mapping to the corresponding word sequence, which is learned automatically.

[5]The robust parser was provided by an industry partner of the NaDia project; details about its realization are not publicly available.

Chapter 4

Integration of speech recognition and interpretation

This chapter presents the detailed description of the efficient implementation of the one-stage decoding algorithm which uses the WTNH representation of semantic grammars introduced in the last chapter in order to determine the best-matching parse tree directly from the recorded speech signal. In contrast to the one-stage decoder implementation described in [Tho06], which used a static search space organization and was intended for carrying out off-line experiments, the efficient implementation of the one-stage decoding algorithm is intended for real time processing of utterances inside a spoken dialog system. Algorithmic efficency plays a crucial role in this context, because the processing of spoken utterances in real-time is a prerequisite for the dialog itself: During conversation, the user expects an immediate reaction of the dialog system, only a short time after finishing his utterance.

The gain in efficency is achieved by organizing the search space dynamically by the use of on-demand operations on finite-state automata. These operations allow the composition and optimization of the integrated search network during the progress of the decoding process. This provides a good trade-off between decoding speed, memory consumption and initialization effort.

The chapter starts with the description of the one-stage decoder implementation with the static search space organization. After a presentation of the calculus on weighted finite-state transducers (WFST) from the perspective of its well-known application to speech recognition, it will be shown how this approach has been used to improve the efficiency of the one-stage decoder.

4.1 Static search space organization

As stated in the last chapter, the representation by means of the weighted transition network hierarchy (WTNH) covers all knowlegde sources that are involved in the speech interpretation task.[1] The uniform representation of the acoustic-phonetic, lexical and semantic-syntactic knowledge provides a straightforward way to determine the parse tree, which best matches the recorded speech signal by searching the

[1]Except for the scripting commands of the semantic grammar which extract the slot-value pairs from a decoded parse tree.

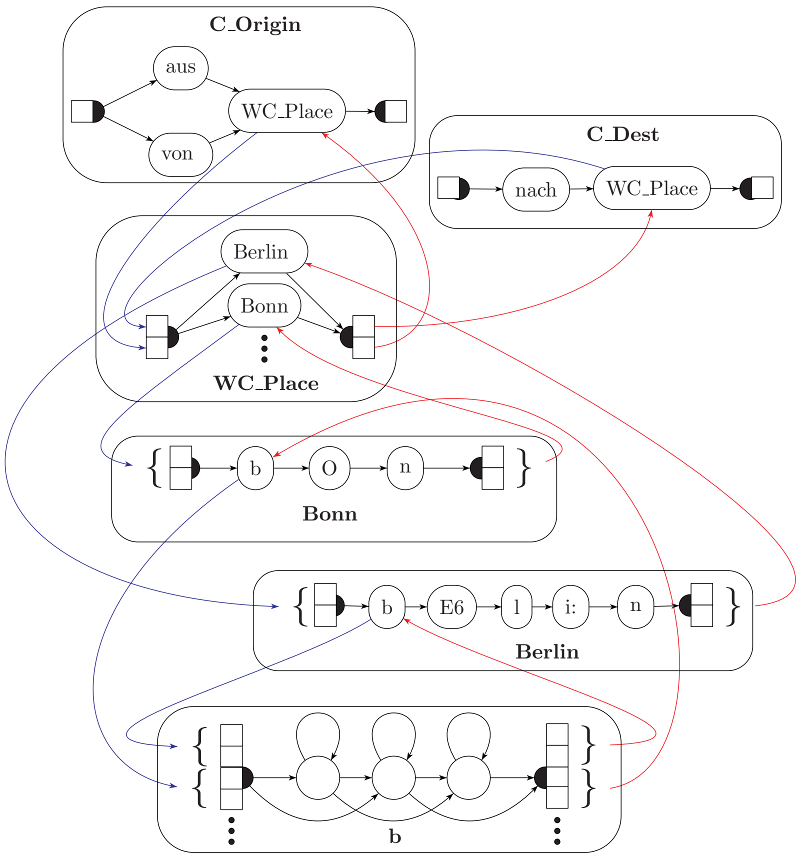

Figure 4.1: Example demonstrating the static token passing strategy for the WTNH which relies on sharing the memory reserved for tokens.

best path through the through the WTNH.

The problem of finding the best path through the WTNH can be solved with the Viterbi decoding algorithm and the token passing principle, which have been presented in section 2.5 in the context of word-based continuous speech recognition. The difference to the simpler case of a word-based language model, is that the WTNH representation generalizes the principle of substitution. When constructing the integrated search network for a word-based language model, there are usually only two levels of substitution: words are replaced by their pronunciation, and phonemes by their corresponding Hidden Markov Models. For determining the best-matching word sequence, only the exit nodes of words have to be recorded during token passing search. The WTNH representation, however, allows arbitrary deeply nested substitutions of network nodes. To reveal the best-matching parse tree, one has to record all nodes that represent symbols of the underlying semantic grammar.

For extending the token passing algorithm to work on the WTNH representation it is necessary to prevent the recombination of tokens which reside in different instances of the same subnet which result from using the same subnet in different contexts. A simple way to achieve this is to duplicate the topology of a particular subnet each time when it is used in some context of another subnet. This approach simply compiles the network hierarchy into a flat network which is processed with the standard token passing search.

The duplication of the network topology on each use of a subnet appears costly, especially if the whole compilation of the WTNH has to be carried out in advance. To prevent this, another possibility is to duplicate only the memory reserved for the tokens, and not the network topology. In this case, the token passing algorithm has to be extended to allow the propagation of tokens from different subnet instances in parallel along the subnet's uniquely stored network topology. This approach has been selected for the first implementation of the one-stage decoder (see [Tho06]).

Tokens which correspond to different instances of the same subnet are separated by their socalled "token history". This is the sequence of passed subnet nodes while walking down the network hierarchy. In the actual implementation of the decoder, the token history is not explicitly stored with each token, but is implicitly represented by sharing the memory reserved for tokens between parent and child networks. The memory reserved for the token collection at a subnet node is also used for the token collection at the entry node of the corresponding child network. Provided that during token propagation child networks are processed after their parent networks, this strategy ensures that all tokens are automatically carried down the network hierarchy until they all reside on the level of Hidden Markov Models which represent the acoustic models of specific phonemes.

After the downward propagation, the evaluation of the emission probability density function at each HMM state considers the current acoustic feature vector which has been extracted from the speech signal. Then, the updated tokens have to be propagated again upwards in the network hierarchy. This is achieved by sharing the memory for tokens between subnet nodes and exit nodes of subnets. All tokens which reside at an exit node of a subnet are also referenced by the corresponding subnet nodes inside the parent networks. By processing the subnets in reverse order, which means child networks before parent networks, all tokens automatically climb up the network hierarchy and are propagated further during the downward propagation phase of the next time step.

Figure 4.1 shows this token passing strategy for a snippet of the WTNH from the airport information domain. A subnet node has references to incoming and outgoing tokens. During downward propagation, incoming tokens are written into a specific location of the token collection at the entry node of the corresponding subnet. On the other hand, outgoing tokens are taken from the same location of the token collection at the exit node. Link nodes which do not refer to a subnet (including entry and exit node) have identical incoming and outgoing tokens. The order in which subnets have to be processed during the token propagation phases results from the topological order of the super network that represents the dependencies between subnets. Inside each subnet, the order in which network nodes are processed is not critical, as long as no tokens remain on a link node which is not the exit node. To prevent this, network nodes are also sorted in topological order if this is possible. For cyclic subnets, which have no topological order, the token propagation has to be carried on until there are no tokens left on link nodes.

The allocation of the necessary token memory and the initialization of the references to incoming and outgoing token collections for every subnet node have to be carried out in advance. This is done by enumerating all contexts in which a particular subnet occurs in the WTNH, which is the number of different paths in the super network that lead from the root subnet to that particular subnet.

It is true that this approach prevents the duplication of the topology of subnets. However, the search space layout can only be optimized locally for each particular subnet. Redundant paths in different subnets cannot be merged together, which is important to achieve a significant increase in decoding speed. Furthermore, the static reservation of the search space does not support recursive relations between subnets.

To achieve an efficient implementation which is suited for the real-time use in a spoken dialog system, the initial one-stage decoder architecture has been redesigned to support the dynamic replacement of subnets and a global search space optimization by means of the weighted finite-state transducer calculus.

4.2 Introduction to WFST-based speech recognition

For an introduction to the theory of weighted finite-state transducers (WFST) and to its application to the problem of the efficient integration of knowledge sources in speech recognition, this section provides a brief summary of the most important publications on this subject (e.g. [MPR02, PR97, Moh97]). This introduction should give the reader the necessary background information to comprehend the application of the WFST framework in the context of this thesis. However, a detailed presentation of the underlying theory of semirings and rational power series is beyond the scope of this work.

The section begins with a discussion about the great impact of the WFST calculus on large vocabulary speech recognition systems.

4.2.1 Efficient integration of knowledge sources in LVSR

Especially speech recognition for large vocabularies (LVSR) requires a decoder architecture which efficiently combines the knowledge sources language model, pronunciation lexicon and acoustic models to build up the search space, in which the best-matching hypothesis is determined by the decoding algorithm. Examples for typical LVSR-techniques using n-gram language models are (see [Ort98]):

- **Tree-structured lexicon.** To reduce the size of the search space the pronunciation lexicon is represented by a prefix tree. The combination with the language model requires the management of various copies of the lexicon tree, one for each n-gram history.

- **Language model look-ahead.** With the use of the lexicon tree, the identity of a word remains undefined until a specific hypothesis has reached a tree leaf. To prevent the late evaluation of the language model, the language model probabilities are distributed over the edges of each lexicon tree copy.

These techniques allow efficient implementations, but lead to very complex decoder architectures, especially when considering cross-word context dependent phone models. Another disadvantage of these implementations is the hard-coded interface with the n-gram language model, which doesn't allow to switch easily to another type of language model.

An alternative approach is the use of weighted finite-state transducers (WFST) and the associated calculus, which has been introduced to the speech recognition community by Mohri, Pereira and Riley (see [MPR02]). This theory provides the foundation for a uniform combination of knowledge sources that can be represented by finite-state autotmata, which is usually the case for the knowledge sources involved in speech recognition. The concatenation of automata operations allows to integrate all knowledge sources into a single finite-state automaton, yielding an optimal representation of the search space, which can be processed very efficiently by time-synchronous Viterbi decoding. The implementation of the decoding algorithm remains quite simple, because the complex integration of the knowledge sources is uniformly handled by the WFST calculus. It turned out that all hard-coded optimizations in conventional LVSR decoders can be realized by corresponding automata operations. Actually, the WFST approach is even more efficient, because it achieves a global optimization of the search space. At the same time, it offers more flexibility concerning the integration of different knowledge sources into the speech recognition system (see [KNRM02]).

What makes the WFST approach even more appealing, is the possibility to carry out the automata operations on-demand. This means that the search space doesn't have to be built up in advance, but can be constructed progressively, driven by the set of active hypotheses which is tracked by the decoding algorithm. The pruning of hypotheses leaves those parts of the search space unconstructed, which badly match the observed speech signal. This saves memory consumption and reduces the time necessary for the initialization phase. Because the construction of the whole search space may lead to very large automata, these advantages are very relevant in practice and justify the loss of decoding speed that is caused by the overhead of the on-demand construction. Furthermore, the on-demand construction of the search space allows to handle knowledge sources, which cannot be represented by a finite number of automaton states, like recursive grammars.

4.2.2 Definition of WFSTs

A finite-state transducer is a finite-state automaton that has input and output labels attached to its transitions.[2] The paths through the transducer map input label sequences to output label sequences. A weighted transducer also includes weights on the transitons, which usually represent probabilities. This allows to compute an overall probability of an input-to-output label mapping along a specific path through the transducer.

The arithmetic used to calculate with weights is defined by the algebraic concept of a "semiring". A semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is defined over the set of weights $\mathbb{K}$ by a logical "plus" operation $\oplus$ and a logical "times" operation $\otimes$ and their corresponding identity weights $\bar{0}$ and $\bar{1}$. These operations are used by the automata operations to calculate sums or products of weights. For the application to speech recognition, the "tropical" semiring $(\mathbb{R}^+ \cup \infty, \min, \cdot, \infty, 0)$ provides an appropriate definition,

[2]Mohri et al. prefer the term "label" instead of "symbol".

because it reflects the representation of probabilities on logarithmic scale and the Viterbi (best path) approximization, which replaces the complex logarithmic sum $-\ln(e^{-a} + e^{-b})$ by the minimum of the two weights a and b.[3]

Formally, a WFST $T = (\Sigma, \Omega, Q, E, i, F, \rho)$ is represented by

- the label sets Σ and Ω of input and output alphabet,
- the set of states Q,
- the set of transitions $E \subset Q \times (\Sigma \cup \epsilon) \times (\Omega \cup \epsilon) \times \mathbb{K} \times Q$, which is a subset of all combinations of source and destination states, input and output labels and weights,
- the initial state $i \in Q$,
- the set of final states $F \subseteq Q$,
- and the final weight function $\rho : F \rightarrow \mathbb{K}$.

In the original definition of [MPR02], there is also an initial weight λ, which is however neglected in practice and thus left apart by setting it always to $\bar{1}$.

A transition $t = (p[t], \ell_i[t], \ell_o[t], w[t], n[t])$ is defined by a source state $p[t]$, a destination state $n[t]$, an input label $\ell_i[t]$, an output label $\ell_o[t]$ and a weight $w[t]$. The label ϵ is a special label which denotes an "empty" label. A "successful" path π is a sequence of consecutive transitions $t_1, \ldots, t_n$, which starts at the initial state i and ends at a final state $f \in F$. The weight $w[\pi]$ of a successful path π is the $\otimes$-product of the transition weights and the final weight $\rho(n[t])$:

$$w[\pi] = w[t_1] \otimes \ldots \otimes w[t_n] \otimes \rho(n[t_n])$$

WFSTs are a generalization of weighted finite-state acceptors (WFSA), that have only a single label attached to each transition. Automata operations defined for WFSTs treat a WFSA like a WFST with identical input and output labels.

In graphical representations of WFSTs, like in figure 4.5, transitions are depicted by arcs[4], which connect source and destination state and carry a concatenated label with the format $\ell_i[t] : \ell_o[t]/w[t]$. The initial state is symbolized by bold print; the final states are symbolized by concentric circles.

4.2.3 Automaton representation of knowledge sources

Interdependent knowledge sources can be represented by transducers mapping from labels on a lower conceptual level to labels on a higher conceptual level. For the common knowledge sources used in speech recognition this is

- an acoustic model transducer H, which maps from a sequence of Hidden Markov Model states to phonemes,

[3]The exotic term "tropical semiring" honors the extensive work of Imre Simon on this subject, which he has done in Brazil.

[4]In the following transitions of finite-state automata are simply called arcs.

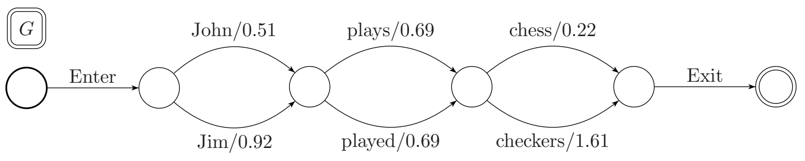

Figure 4.2: “Toy” language model represented by the weighted acceptor G.

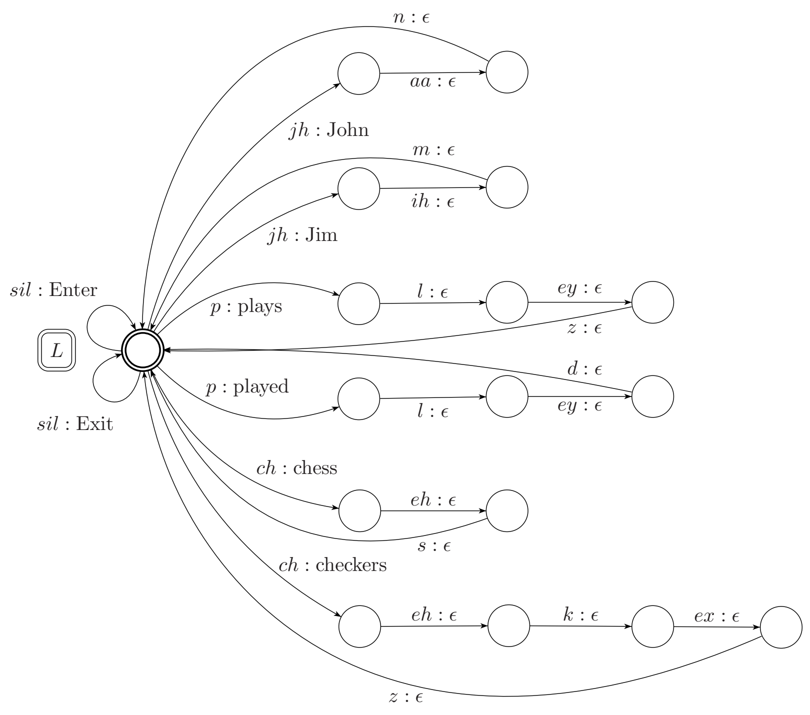

Figure 4.3: Phonetic lexicon for language model G represented by transducer L.

- a pronunciation lexicon transducer L which maps from a sequence of phonemes to words.
- and a weighted acceptor G representing the language model, which assigns weights to possible word sequences.

In the case of context-dependent phoneme models there is also a transducer C which maps context-dependent phonemes to simple phonemes.

A “toy” example of a language model acceptor and the corresponding lexicon transducer is shown in figures 4.2 and 4.3.

4.2.4 Composition and Determinization

There are several operations defined on WFSTs, but the most important ones, concerning the intended integration of knowledge sources are “composition” and “determinization”. The composition operation allows to combine two automata which represent interdependent knowledge sources like language model and pronunciation lexicon. The determinization operation is used to transform an automaton into an equivalent deterministic automaton, where each state has outgoing arcs with unique input labels. Determinization is usually applied to the outcome of a composition operation to remove redundant paths, which causes the speedup of the decoding process. Therefore, the concatenation of composition and determinization operations permits a standardization of the network topology of the search space, which significantly reduces the necessary number of states and arcs.

The operation "weighted minimization", which is actually a combination of the "weight pushing" and the "minimization" operation, allows even further optimization of the search space representation (see [Moh97, MR01]). However, since the software library [KN04] which has been used in the context of this thesis doesn't include on-demand implementations of the corresponding algorithms, the "weighted minimization" operation has not been exploited in the presented work.

Composition

The composition operation, which is denoted by the symbol "$\circ$", combines two weighted transducers $T_1(\Sigma, \Gamma, Q_{T_1}, \ldots)$ and $T_2(\Gamma, \Delta, Q_{T_2}, \ldots)$ and yields the weighted transducer $T(\Sigma, \Delta, Q_T, \ldots)$. It maps label sequences in the input alphabet Σ of the first transducer T_1 to label sequences in the output alphabet Δ of the second transducer T_2.

Figure 4.4 shows the transducer resulting from the composition of the example automata L and G, representing pronunciation lexicon and language model in speech recognition. This transducer maps from phoneme sequences to weighted word sequences, which follow the language model. By another composition with the acoustic model transducer H, the resulting cascade of compositions $H \circ L \circ G$ maps from HMM state sequences to word sequences and represents the final search space on which operates the Viterbi decoder.

At first glance, the composition operation has the same effect like a context-free substitution of arcs by corresponding sub-automata. For the simple example of figure 4.4 this is true, but the difference between the two operations is that composition involves only two automata, whereas the substitution operation needs a separate automaton representation for each label to replace. Thus, composition operates in a more "compact" fashion than substitution. Moreover, the composition operation is able to carry out context-sensitive substitutions inside label sequences represented by arbitrary automata, which no longer can be achieved by simple context-free substitutions. An application for this kind of composition is the context-dependent phoneme transducer C, which is composed with $L \circ G$ to generate a transducer with context-dependent phoneme labels on the input side (see section 4.3.4).

The pseudo-code fragment 4.1 and figure 4.5 illustrate what actually happens during the composition of two ϵ-free WFSTs, $T = T_1 \circ T_2$. The set of states Q_T of the composed transducer correspond to pairs of states from left and right operand: $Q_T \subseteq Q_{T_1} \times Q_{T_2}$. The composition starts from the pair of initial states (i_{T_1}, i_{T_2}). If the current state pair is (q_1, q_2), a new pair of states $(n[t_1], n[t_2])$ is added to the state set if there is an arc from q_1 to $n[t_1]$ in T_1 and an arc q_2 to $n[t_2]$ in T_2 which have matching input and output labels $\ell_o[t_1] = \ell_i[t_2]$. The new arc which connects (q_1, q_2) and $(n[t_1], n[t_2])$ in the composed transducer carries the input label $\ell_i[t_1]$ of

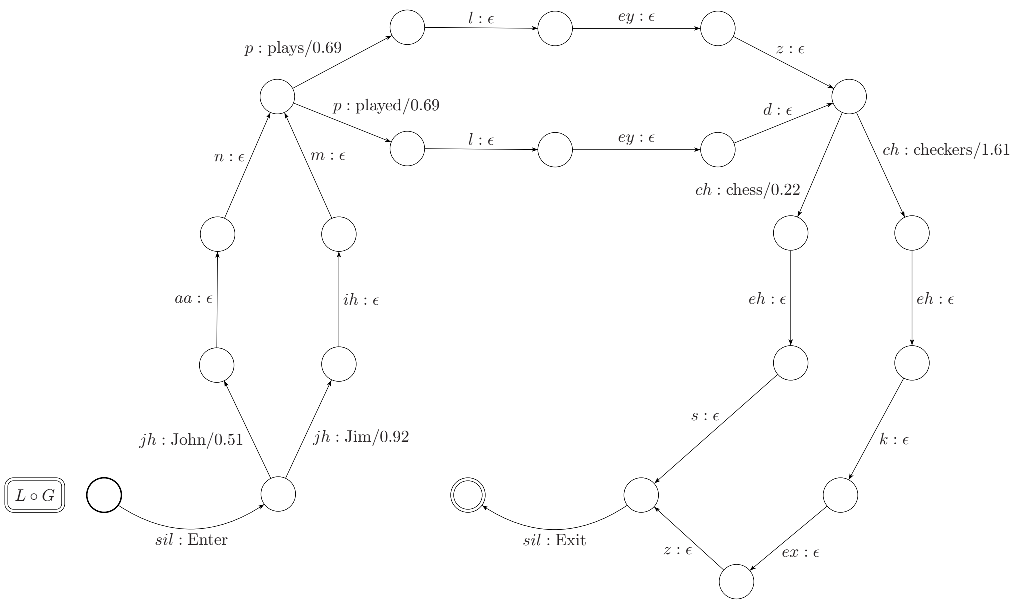

Figure 4.4: Composition of “toy” lexicon L and language model G.

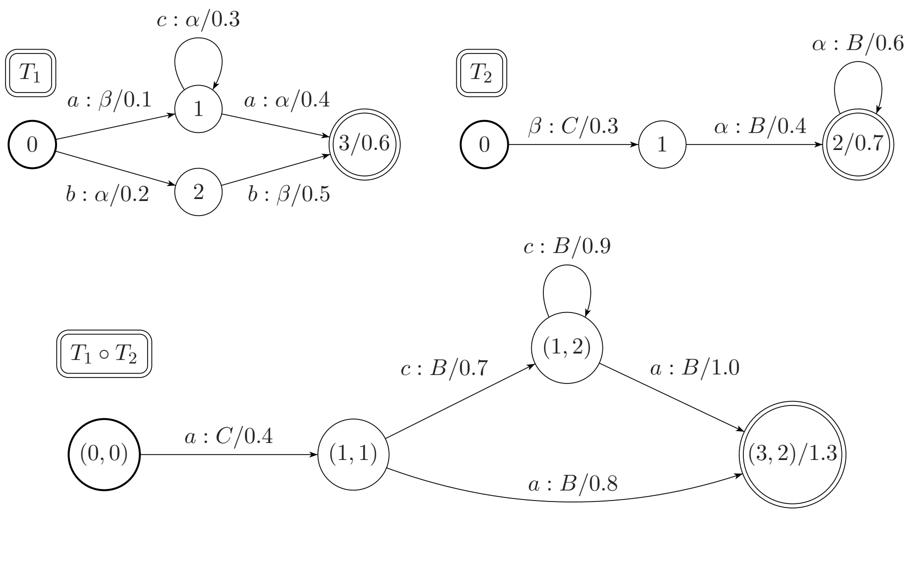

Figure 4.5: Example from [MPR02] for the composition of ϵ-free WFSTs.

Pseudo-code fragment 4.1 Composition of two ϵ-free transducers, $T = T_1 \circ T_2$. S represents a stack, which stores pairs of state indices (see [KN04]).

```
$i_T \leftarrow (i_{T_1}, i_{T_2})$                    ▷ Initialize stack with pair of initial states.
$S \leftarrow i_T$
while $S \neq \emptyset$ do                             ▷ Do while stack is not empty.
    $(q_1, q_2) \leftarrow S$                            ▷ Pop next state pair $(q_1, q_2)$ from stack.
    $Q_T \leftarrow Q_T \cup (q_1, q_2)$        ▷ Add state pair to state set of composed trasducer.
    if $q_1 \in F_{T_1} \wedge q_2 \in F_{T_2}$ then                          ▷ Both final states?
        $F_T \leftarrow F_T \cup (q_1, q_2)$               ▷ Yes, add state pair to final state set.
        $\rho_T((q_1, q_2)) \leftarrow \rho_{T_1}(q_1) \otimes \rho_{T_2}(q_2)$        ▷ Calculate final weight product.
    end if
    for each $(p[t_1], \ell_i[t_1], \ell_o[t_1], n[t_1]) \in E_{T_1}, p[t_1] = q_1$ do
                                                       ▷ For each outgoing arc of state $q_1$.
        for each $(p[t_2], \ell_i[t_2], \ell_o[t_2], n[t_2]) \in E_{T_2}, p[t_2] = q_2 \wedge \ell_o[t_1] = \ell_i[t_2]$ do
                              ▷ For each outgoing arc of state $q_2$ with matching output label.
            $E_T \leftarrow E_T \cup ((p[t_1], p[t_2]), \ell_i[t_1], \ell_o[t_2], w[t_1] \otimes w[t_2], (n[t_1], n[t_2]))$
                         ▷ Add arc with corresponding labels and weight product to arc set.
            if $(n[t_1], n[t_2]) \notin Q_T$ then
                         ▷ Is destination state pair a new state of composed transducer?
                $S \leftarrow (n[t_1], n[t_2])$                    ▷ Yes, push state pair on stack.
            end if
        end for
    end for
end while
```

the arc in T_1 and the output label $\ell_o[t_2]$ of the arc in T_2. Its weight is the $\otimes$-product of the corresponding arc weights $w[t_1]$ and $w[t_2]$. Newly discovered state pairs are pushed onto a stack S, which is processed until all final composed states are reached.

The composition of WFSTs with ϵ labels, which is usually the case in practice, is more complex and requires an intermediate composition with a "filter" transducer (see [PR97]). This intermediate composition can be realized in an "unrolled" fashion, which allows efficient implementations of the general composition algorithm, like those in [MPR98, KN04, Het04].

Determinization

The transducer $L \circ G$ of figure 4.4 has various states which have outgoing arcs that carry the same input label. The determinization operation, which is denoted by *det*(), merges this states and redistributes weights and output labels. This yields an equivalent deterministic transducer, which only has states that have outgoing edges with unique input labels.

Figure 4.6 shows the transducer resulting from $det(L \circ G)$ if assuming the tropical semiring to combine the involved weights. The resulting transducer provides an optimized version of the original transducer which represents the same weighted input-to-output label mapping with less states and arcs. The determinization operation has a similar effect like the construction of lexicon trees in conventional LVSR

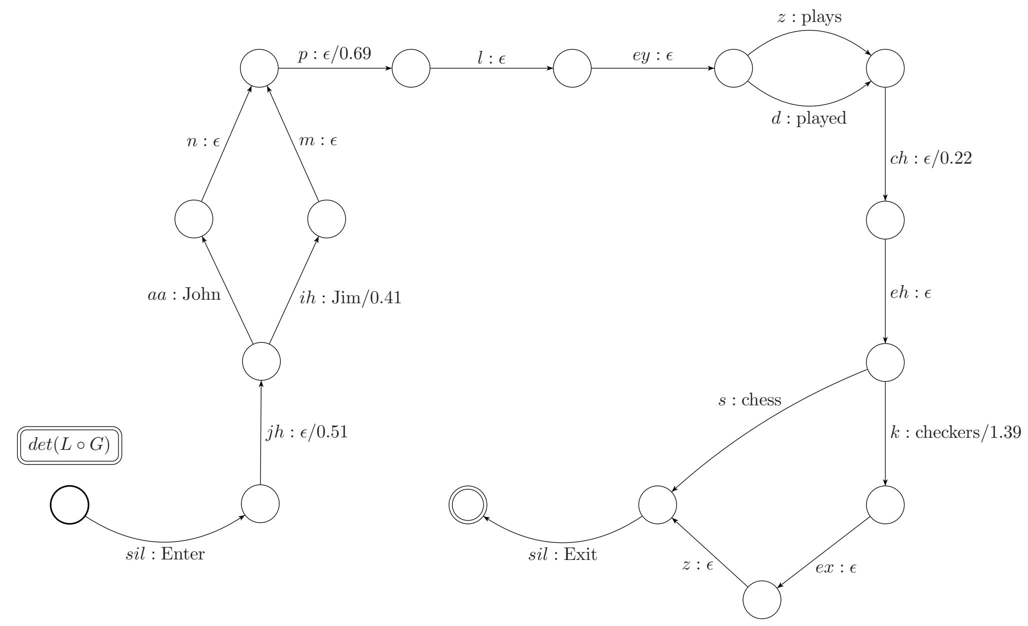

Figure 4.6: Optimized representation of $L \circ G$ resulting from determinization.

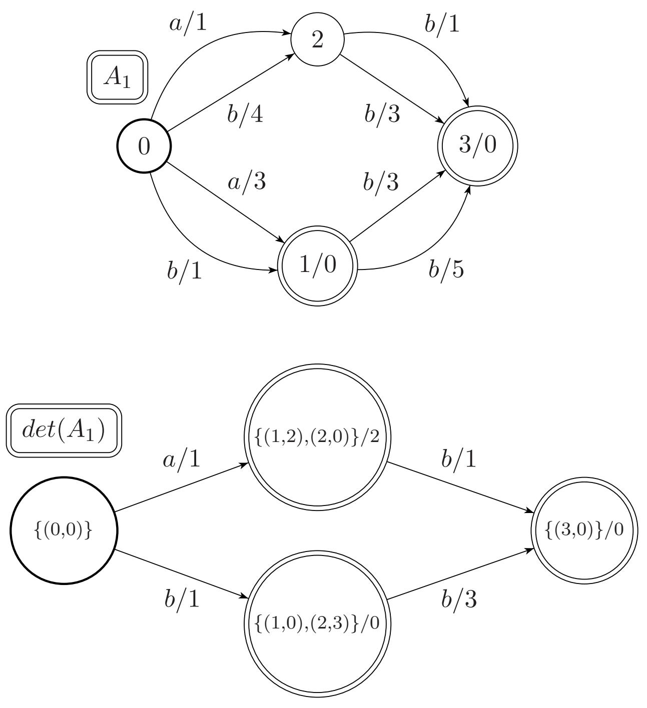

Figure 4.7: Example from [Moh97] for the determinization of WFSAs.

decoder implementations, but works in a much more general manner. It generates partial lexicon tree copies which individually fit into each context of an arbitrary language model given by a finite-state automaton.

The algorithm realizing the determinization operation is based on the construction of subsets, which include states that are reachable from the initial state by the same input label sequence (see [Moh97]). Similar to the state pairs in the composition algorithm, subsets represent “abstract” states of the determinized automaton, which are accessed in practical implementations of the algorithm by sophisticated hashing techniques.

For the sake of clarity, this section only presents the determinization algorithm for the case of weighted acceptors. An example for $A_2 = det(A_1)$, assuming the tropical semiring, is depicted in figure 4.7; the algorithm is given by the pseudo-code fragment 4.2. A state of the determinized acceptor A_2 corresponds to a subset, which is a set of pairs (q_1, x) consisting of a state of the original acceptor A_1 and a residual weight x. Residual weights are used to redistribute weights when merging states that can be reached by same input label sequence. The construction of the determinized acceptor starts with the initial subset $\{(i_1, \bar{1})\}$. For each label a which occurs on arcs that leave states q_1 in the current subset q_2, a new subset is constructed in the following way: First one has to determine the weight $w[t_2]$ of the arc leading to the state in the determinized acceptor A_2 that corresponds to the subset which is to be constructed. The weight $w[t_2]$ is calculated by summing all products of residual weights x corresponding to states q_1 with the sum over all weights $w[t_1]$ on outgoing arcs of q_1 with label a. Then, by the aid of the inverse weight $(w[t_2])^{-1}$ the residual weights of the new subset can be calculated. The inverse weight satisfies $(w[t_2])^{-1} \otimes w[t_2] = \bar{1}$; in the case of the tropical semiring this is simply the negative weight $-w[t_2]$. The new subset consists of states q_1', which at least have one incoming arc with label a from a source state $p[t_1]$ in the current subset q_2. The residual weights corresponding to the states q_1' are calculated by a sum over products. Each product includes the inverse weight $(w[t_2])^{-1}$, the residual weight x corresponding to the source state $p[t_1]$, and the arc weight $w[t_1]$. The sum is carried out over all arcs with label a which lead to q_1' and come from a source state $p[t_1]$ in the current subset q_2. Thus, the inverse weight $(w[t_2])^{-1}$ liberates the residual weights from the contribution of the weight of the newly created arc with label a, which leads to the state corresponding to the constructed subset. If this subset has never been constructed before, it is pushed onto the stack S, which is processed until all final determinized states are reached.

In the more general case of determinization of weighted transducers, also output labels have to be redistributed in the resulting transducer. To achieve this, in addition to residual weights, the extended algorithm also considers residual output label sequences, which are determined by the aid of the least common prefix operation defined on label sequences. Efficient implementations of this extended algorithm get quite complex, especially because of the sophisticated hashing techniques, which are necessary for a quick access of the abstract states, which now represent sets of triples including a state, a residual weight and a residual output label sequence (for further details, see published implementations of [KN04] and [Het04]).

Pseudo-code fragment 4.2 Determinization of weighted acceptor, $A_2 = det(A_1)$. S represents a stack storing subsets, which contain pairs of states and residual weights (see [Moh97]).

```
$i_{A_2} \leftarrow \{(i_{A_1}, \bar{1})\}$                    ▷ Initialize stack with initial subset pair.
$S \leftarrow i_{A_2}$
while $S \neq \emptyset$ do                                   ▷ Do while stack is not empty.
    $q_2 \leftarrow S$                                        ▷ Pop next subset from stack.
    $Q_{A_2} \leftarrow Q_{A_2} \cup q_2$           ▷ Add subset to state set of determinized acceptor.
    if $(q_1, x) \in q_2 \wedge q_1 \in F_{A_1}$ then              ▷ Subset contains a final state?
        $F_{A_2} \leftarrow F_{A_2} \cup q_2$                  ▷ Yes, add subset to set of final states.
        $\rho_{A_2}(q_2) \leftarrow \bigoplus_{(q_1,x) \in q_2 \wedge q_1 \in F_{A_1}} x \otimes \rho_{A_1}(q_1)$
                                            ▷ The sum is calculated over all final subset states.
    end if
    for each $a \in \Sigma_{A_1}, (p[t_1], a, w[t_1], n[t_1]) \in E_{A_1} \wedge (p[t_1], x) \in q_2$ do
                                     ▷ For each label "a" on outgoing arcs of subset states.
        $w[t_2] \leftarrow \bar{0}$                                  ▷ Initialize new arc weight.
        for each $(q_1, x) \in q_2, (q_1, a, w[t_1], n[t_1]) \in E_{A_1}$ do
                         ▷ For each subset pair that has an outgoing arc with label "a".
            $w[t_2] \leftarrow w[t_2] \oplus \left( x \otimes \bigoplus_{(q_1,a,w[t_1],n[t_1]) \in E_{A_1}} w[t_1] \right)$ ▷ The sum is calculated
over all outgoing arcs with label "a", leaving the subset state.
        end for
        $n[t_2] \leftarrow \emptyset$                          ▷ Initialize new destination subset.
        for each $q'_1 \in Q_{A_1}, (p[t_1], a, w[t_1], q'_1) \in E_{A_1} \wedge (p[t_1], x) \in q_2$ do
         ▷ For each destination state of arcs with label "a" leaving a subset state.
            $n[t_2] \leftarrow n[t_2] \cup \left\{ \left( q'_1, \bigoplus_{(p[t_1],a,w[t_1],q'_1) \in E_{A_1} \wedge (p[t_1],x) \in q_2} (w[t_2])^{-1} \otimes x \otimes w[t_1] \right) \right\}$
    ▷ The sum is calculated over all incoming arcs which carry the label "a" and
leave a subset state.
        end for
        $E_{A_2} \leftarrow E_{A_2} \cup (q_2, a, w[t_2], n[t_2])$          ▷ Add arc with label "a" to arc set.
        if $n[t_2] \notin Q_{A_2}$ then
                              ▷ Is destination subset new state of determinized acceptor?
            $S \leftarrow n[t_2]$                                  ▷ Yes, push subset on stack.
        end if
    end for
end while
```

4.2.5 Problem of "determinizability"

Unfortunately, the determinization algorithm does not terminate for arbitrary input automata. In the case of transducers, this happens if there are paths with the same input label sequence that map to different sequences of output labels. This problem is not restricted to artificially constructed examples, but occurs quite frequently in practical applications: the determinization of the combined transducer of phonetic lexicon and language model $L \circ G$ may fail for the case that the lexicon contains homophones, which are words with different orthography but equal pronunciation.

A possible approach to prevent the infinite subset construction inside the determinization algorithm is to insert arcs carrying special "disambiguation" labels at adequate locations in the input transducer. After successfull determinization, these labels are replaced by ϵ labels. Corresponding arcs may be deleted by the application of an operation called "ϵ-removal" (see [Moh02]).

For the special case of the lexicon transducer, the disambiguation labels can be inserted on arcs which return from the last state of a path representing a specific word back to the initial state. So, disambiguation labels correspond to special phonemes appended to the end of the phonetic transcription of each word. The simplest solution is to use the word label itself as disambiguation label. Actually, one needs less different disambiguation labels: only as much as there are homophones plus another one marking the end of a word (see [MPR02]). However, this optimal strategy works only for the special case of phonetic lexicon transducers. A general algorithm which can render arbitrary automata "determinizable" with a minimum number of modifications of the input automaton has been presented by [AM03]. However, probably due to its high complexity, this algorithm is not part of the open-source implementations [KN04, Het04].

Therefore, in the context of the decoder implementation which accompanies this thesis, the problem of determinzability has been solved by copying each output label as disambiguation label on the input side of a corresponding transducer arc. This simple method to allocate and distribute disambiguation labels produces more computational overhead than actually necessary. However, because of the moderate vocabulary sizes of the semantic grammars used for the intended speech interpretation task, the overhead produced by the disambiguation labels does not have a considerable impact on the efficiency of the decoding process. The details on how to use output labels for disambiguation are explained later on, when presenting the exact layouts of the automata wich represent the knowledge sources processed by the one-stage decoder that integrates speech recognition and parsing (see section 4.3).

4.2.6 On-demand computation of local automata operations

An interesting property of the composition and the determinization operation is the fact that they can be carried out "locally". The creation of an output state depends only on input states which are included in the corresponding abstract state. Creating a new output state actually means creating its outgoing arcs which involves the allocation of the abstract states corresponding to their destination states. Abstract states are allocated beginning from the initial one that only includes initial states of the input automata. Thus, the result of a local operation can be constructed

Pseudo-code fragment 4.3 Interface of the abstract base class for automata, that allow the on-demand computation of local automata operations (see [KN04]).

```
class AUTOMATON
    class ARC
        StateId TARGET()                          ▷ Returns target state id.
        Weight WEIGHT()
        LabelId INPUT()
        LabelId OUTPUT()
    end class
    class STATE
        StateId ID()
        Weight FINAL()                            ▷ Returns final weight.
        ARCITERATOR BEGIN()      ▷ Iterators for accessing outgoing arcs.
        ARCITERATOR END()
    end class
    StateId INITITALSTATEID()        ▷ Returns the id of the initial state.
    STATE GETSTATE(StateId)  ▷ Returns the state instance corresponding to the
passed state id.
end class
```

progressively starting from the initial states of the input automata. The power of the progressive construction becomes obvious when concatenating local automata operations. This allows to access states of the automaton which results from the concatenation of several operations, whithout having to construct the intermediate automata entirely. The states of the intermediate automata are constructed on-demand while traversing the final automaton.

The software design which realizes the on-demand computation of local automata operations has been introduced by [MPR98]. Expressed in terms of object-orientated programming, this concept relies on an abstract base class for automata providing the interface which is shown in the pseudo-code fragment 4.3. Classes derived from the abstract base class AUTOMATON represent either automata resulting from a specific local operation, or the static automaton which is entirely present in memory. A class representing a specific automata operation accesses the input automata of the operation via the AUTOMATON interface. States and their corresponding abstract states are accessed by integer identifiers which are denoted by "StateId". The outgoing arcs of a specific state can be enumerated by corresponding iterators provided by the interface class STATE. Instances of the class ARC store the state identifier of the arc's destination state, which is accessed by the TARGET() member function.

The member function INITIALSTATEID() of the AUTOMATON class allocates the initial abstract state and returns its state identifier. Output states resulting from a specific automata operation are created by the member function GETSTATE(). The passed state identifier refers to the abstract state from which the on-demand construction is spanned further. The abstract state corresponds to the newly created state which is returned by the GETSTATE() member function. The generation of its outgoing arcs includes the allocation of abstract states which correspond to

Pseudo-code fragment 4.4 Application of the on-demand construction of the determinized composition $det(L \circ G)$ of phonetic lexicon and language model.

```
function AUTOMATON COMPOSE(AUTOMATON a1, AUTOMATON a2)
    return new ComposeAutomaton(a1, a2)
end function

function AUTOMATON DETERMINIZE(AUTOMATON a)
    return new DeterminizeAutomaton(a1)
end function
...

                    ▷ Load lexicon and language model as static automaton into memory.
lex ← new STATICAUTOMATON()
lm ← new STATICAUTOMATON()
READ(lex, file1)
READ(lm, file2)

                ▷ Creates the automaton resulting from determinization and composition,
without actually creating its states.
detLexLm ← DETERMINIZE(COMPOSE(lex, lm))

                                              ▷ Begin with traversal of resulting automaton
stateId ← detLexLm.INITIALSTATEID()
state ← detLexLm.GETSTATE(initialId)
for each arc of state do
    ...
end for
```

their destination states. The allocation of abstract states again involves calls to the GETSTATE() member function of the input automata. This triggers the progressive construction of intermediate states of input automata that are themselves constructed on-demand. The chain of recursive calls ends when calling the GETSTATE() member function of a static automaton, which usally represents a specific knowledge source loaded entirely into memory.

The pseudo-code fragment 4.4 shows the application of the on-demand construction for the example of figure 4.6 which reflects the determinized composition $det(L \circ G)$ of phonetic lexicon L and language model G. The functions COMPOSE() and DETERMINIZE() return automata which provide the corresponding on-demand implementations of these operations by their GETSTATE() method. These functions are used to create the automaton which results from the concatenation of the determinization and the composition operation that is effected on the static automata representing phonetic lexicon and language model. The states of this automaton are constructed on-demand during its traversal, which starts from the initial state. In practice, also the other knowledge sources necessary for speech recognition are integrated in this fashion. This enables the on-demand construction of the final search space while traversing it under the control of the Viterbi decoder.

As mentioned before, the software design for the on-demand computation of local automata operations has been introduced by Mohri, Pereira and Riley (see [MPR98]). However, their software library has been made publilcly available only on the level of command line tools, without providing an interface for the on-demand computation. Finally, after a couple of years, other scientists published their own source code of efficiently implemented on-demand algorithms (see [KN04, Het04]). These libraries don't cover the whole functionality of the current library version of [MPR98], which still remains unpublished, but include the most important algorithms, like composition and determinization. In the context of this thesis, the library of [KN04] has successfully been used to achieve an efficient implementation of the one-stage decoder which operates on stochastic context-free grammars.

4.3 WFST-based integration of recognition and parsing

This section focuses on how the WFST approach has been applied to integrate speech recognition and parsing on the basis of stochastic context-free grammars. This is achieved by a search space layout that preserves the hierarchical relations between terminal and non-terminal grammar symbols. By using Viterbi decoding, this allows to determine the best-matching parse tree directly from the speech signal. During the decoding process, the search space is constructed on-demand by the concatenation of WFST operations which act on the automata that represent the involved knowledge sources, like stochastic context free grammar, phonetic lexicon and acoustic model.

As discussed in chapter 3, a stochastic context free grammar used for semantic interpretation is represented by a corresponding hierarchical language model (HLM) which is a hierarchy of weighted transition networks (WTNH). The following subsections present the exact layout of the finite-state automata for the semantic-syntactic, lexical and acoustic-phonetic knowledge sources which will be combined using the WFST approach, beginning with the automaton that represents the HLM.

4.3.1 Weighted finite-state acceptor representing the HLM

In order to integrate a stochastic context-free grammar into the WFST decoding framework, its representation by the corresponding HLM has to be transformed into a corresponding weighted finite-state automaton. Figure 4.8 and 4.9 present an example[5] which shows the relation between both representations.

In the HLM of figure 4.8 the non-terminal subnet nodes A_i refer to subnets with the same label. The subnet nodes a_i are terminal nodes which represent specific words. Subnet edges carry the weights w_i. By definition every subnet must have a single entry and a single exit node[6], which are depicted by black circles.

Figure 4.9 shows the corresponding weighted finite-state automaton $\tilde{G}$, which is a WFSA since input and output labels are identical. The alphabet includes all labels of terminal and non-terminal subnet nodes in the HLM. The automaton provides a "flat" representation of the HLM, which means that non-terminal subnet nodes are

[5]This is a simplistic example with the only purpose to explain the transformation of the HLM into a single weighted finite-state automaton; it doesn't make much sense in practice.

[6]To distinguish clearly between subnets and finite-state automata, the terms "node" and "edge" correspond to subnets, while "state" and "arc" correspond to finite-state automata.

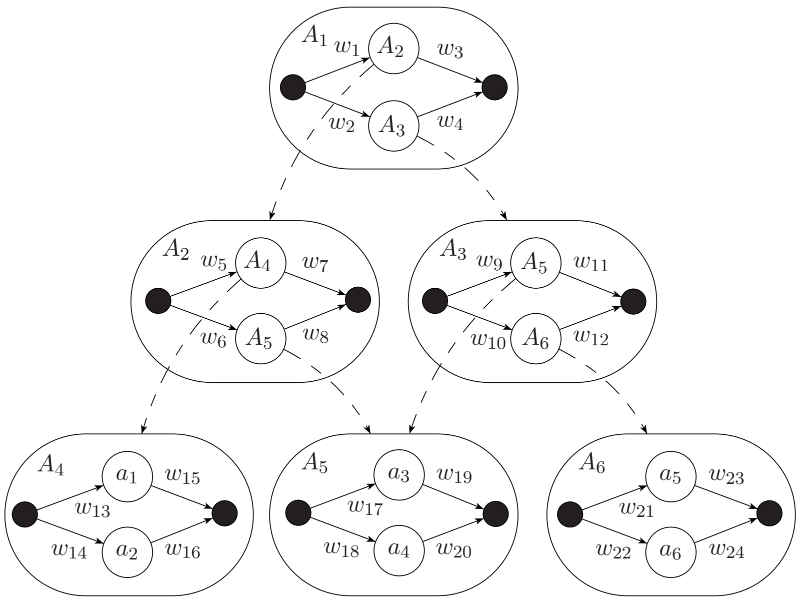

Figure 4.8: Example for a stochastic context-free grammar represented as hierarchical language model (HLM).

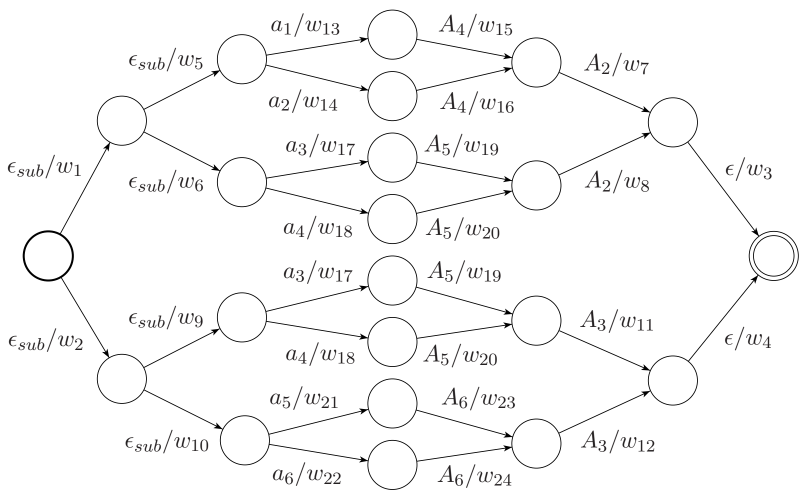

Figure 4.9: Representation of the HLM as weighted finite-state acceptor $\tilde{G}$.

Pseudo-code fragment 4.5 Gets automaton state id which corresponds to a subnet node in a specific network context (helper function).

```
function HlmFsa.getStateId(hierStates, parentCtx, node)
    stateId ← parentCtx[node.getIndex()]
    if stateId = undef then    ▷ Node index not found in parent context hash?
        stateId ← hierStates.getSize()    ▷ Yes, allocate next automaton state id.
        childCtx ← undef
        if node.isNonTerm() then    ▷ Is node non-terminal?
            childCtx ← new NetCtx(stateId)    ▷ Yes, create new child context.
            childCtx[0] ← stateId    ▷ Add entry node index to child context hash.
        end if
        parentCtx[node.getIndex()] ← stateId    ▷ Update parent context hash.
        hierStates[stateId] ← new HierState(parentCtx, childCtx, node)
                ▷ Create hierarchical state saving network contexts and subnet node.
    end if
    return stateId    ▷ Return automaton state id.
end function
```

recursively replaced by instances of the corresponding subnets.[7] Node labels in the HLM become arc labels in the finite-state automaton, which reflects the conversion from Moore to Mealy automaton representation. Each subnet instance in the finite-state automaton starts with an arc carrying the special "opening" label ϵ_{sub} and ends with arcs carrying the "closing" label, which is the label of the subnet itself. However, in the top-level subnet instance the closing label is replaced by the label ϵ because there is no corresponding opening label ϵ_{sub}. This kind of "bracketing" later allows the construction of the parse tree from the best matching path which is determined by the Viterbi decoder.

The relations between automaton states and subnet nodes are the following:

- The destination state of an arc carrying the opening label ϵ_{sub} corresponds to the entry node of a subnet.
- The destination state of an arc carrying a terminal label simply corresponds to the terminal subnet node.
- The destination state of an arc carrying a closing label corresponds to the exit node of the subnet, as well as to the parent subnet node, which refers to the subnet instance.

Subnets may also contain "link" nodes that don't refer to specific words or subnets, just like entry and exit nodes. They serve for saving space when interconnecting several nodes.[8] When building the HLM acceptor, link nodes are treated just like terminal subnet nodes. However, in contrast to the terminal labels which are later expanded by the pronunciation of the corresponding words, the link node label is

[7]The composition operation has not been used, since it doesn't serve well for the successive replacement of specific labels; it's convenient when all labels have to be replaced a single time, like in the case of the composition with the phonetic lexicon transducer.

[8]Link nodes appear for example in the automaton representation of a backoff n-gram model.

Pseudo-code fragment 4.6 On-the-fly generation of automaton representing the hierarchical language model (HLM).

```
function HlmFsa.getState(hierStates, stateId)
    state ← new State(stateId)                              ▷ Create new automaton state.
    node ← hierStates[stateId].getSubNetNode()
                  ▷ Get current subnet node from corresponding hierarchical state.
    netCtx ← hierStates[stateId].getParentCtx()             ▷ Assume terminal node.
    if node.isNonTerm() then                                ▷ Is node non-terminal?
        netCtx ← hierStates[stateId].getChildCtx()          ▷ Yes, get child context.
        subNet ← node.getSubNet()
        node ← subNet.getNode(0)                            ▷ Set node to entry node of subnet.
    else if node.isExitNode() then                          ▷ Is node exit node of subnet?
        stateId ← netCtx.getStateId()                       ▷ Yes, get state id of parent context.
        if stateId = 0 then                                 ▷ Is current subnet on top-level?
            state.setFinal()                                ▷ Yes, set final state.
        else
          ▷ No, get parent node and context from corresponding hierarchical state.
            node ← hierStates[stateId].getSubNetNode()
            netCtx ← hierStates[stateId].getParentCtx()
        end if
    end if
    for each edge of node do                                ▷ For each outgoing edge of current node.
        destNode ← edge.getNode()
        label ← destNode.getLabel()
                  ▷ Assume terminal destination node and set arc label to node label.
        if destNode.isNonTerm() then                        ▷ Is destination node non-terminal?
            label ← $\epsilon_{sub}$                        ▷ Yes, set arc label to opening label $\epsilon_{sub}$.
        else if destNode.isExitNode() then                  ▷ Is destination node exit node?
            if netCtx.getStateId() = 0 then                 ▷ Is current subnet on top-level?
                label ← $\epsilon$                          ▷ Yes, set arc label to $\epsilon$.
            else
                subNet ← node.getNetwork()
                label ← subNet.getLabel()                   ▷ No, set arc label to subnet label.
            end if
        end if
        destStateId ← HlmFsa.getStateId(netCtx, destNode)
                                  ▷ Call helper function to get destination state id.
        state.addArc(destStateId, edge.getWeight(), label)
                                  ▷ Create new automaton arc with edge weight.
    end for
    return state                                            ▷ Return newly created automaton state.
end function
```

Pseudo-code fragment 4.7 Returns id of initial state of the HLM automaton.

```
function HlmFsa.initialStateId(hierStates, topLevelNet)
    initNode ← topLevelNet.getNode(0)    ▷ Get entry node of top-level subnet.
    netCtx ← undef
    if hierStates.getSize() = 0 then    ▷ Called for the first time?
        netCtx ← NetCtx(0)    ▷ Yes, create new network context.
    else
        netCtx ← hierStates[0].getParentCtx()
                                        ▷ No, get from hierarchical state.
    end if
    return HlmFsa.getStateId(hierStates, netCtx, initNode)
                ▷ Call helper function which returns zero for initial state id.
end function
```

treated as a disambiguation label, since its occurrence may distinguish ambiguous paths, that would cause problems during subsequent determinization operations (see section 4.3.2).

The whole finite-state automaton can be created at once. But the resulting automaton may become very large and in the case that the HLM represents a recursive grammar this is even impossible, because the automaton would have an infinite number of states. Thus, the principle of on-demand construction, which has been introduced in section 4.2.6, is applied to the HLM acceptor as well: Given a state identifier, the corresponding state with all outgoing arcs is created on-demand. The creation of an automaton arc doesn't include the creation of the destination state, only the allocation of the corresponding state identifier. The pseudo-code fragment 4.6 shows the implementation of the function getState(), which creates states of the HLM acceptor following the principle of on-demand construction. It is a member function of the class HlmFsa, which represents the HLM acceptor and implements the abstract base class Automaton (see pseudo-code fragment 4.3).

The linkage between automaton states and subnet nodes is managed by the classes HierState and NetCtx. The member variable "hierStates" of the HlmFsa class stores a hierarchical state for every created automaton state. A hierarchical state, which follows the concept of an "abstract state", contains a reference to the corresponding subnet node inside the HLM and references to the parent and child network context. A network context (class NetCtx) stores the mapping between subnet node indices and allocated automaton state identifiers for a specific subnet instance, as well as the state identifier which corresponds to the subnet node in the parent subnet instance. This parent state identifier allows to "return" to the parent subnet instance when reaching an automaton state corresponding to an exit node.

The helper function getStateId(), which is shown in the pseudo-code fragment 4.5, gets the automaton state identifier which corresponds to a subnet node in a specific network context. If not already done, this function allocates an automaton state identifier and creates a new hierarchical state. If the passed subnet node is non-terminal, it creates a child network context, which allows to "enter" the subnet instance when actually creating the automaton state.

The helper function is used by getState() to allocate the state identifiers which correspond to the destination subnet nodes of the outgoing edges of the current subnet node. Automaton arcs receive the weight of the corresponding subnet edge. The arc label results from the kind of the edge's destination subnet node:

- **Non-terminal subnet node.** Entering a subnet instance is indicated with the opening label ϵ_{sub}.
- **Terminal subnet node.** In this case the label simply receives the label of the terminal subnet node.
- **Exit node.** If the current subnet instance is not on top-level, the arc label is set to the closing label which is the name of the subnet. In the top-level subnet the closing label is set to the label ϵ.

The member function INITIALSTATEID() returns the initial state of the HLM acceptor, which results from the passed "top-level" subnet, from which the on-demand construction starts.

Although corresponding to an automaton with an unbounded number of states, recursive grammars can be handled thanks to the on-demand construction: due to the pruning of hypothesises, the decoding algorithm doesn't need to traverse the whole automaton, which limits the number of constructed states. However, since the used Viterbi decoding principle works with the "breath first" strategy, it doesn't allow left recursion. This means, that a recursion cycle in the HLM must contain at least one terminal subnet node, which refers to a lexical entry that consumes time frames (acoustic feature vectors) in its subordinate acoustic models.[9]

4.3.2 Lexicon transducer

The phonetic lexicon transducer $\tilde{L}$ which suits the HLM acceptor $\tilde{G}$ with respect to their composition is shown in figure 4.10. Every terminal label a_i appears on the output side of an arc which leads from the initial state into a branch of the automaton that represents possible pronunciations of the corresponding word. Pronunciations are given by input label sequences over the phonemes p_i that result from paths through that automaton branch. The layout of the lexicon transducer is similar to the one of the "toy" example of figure 4.3. However, there are some extensions which are necessary to keep the result of the composition always determinizable. The self-loops at the inital state, which carry the non-terminal labels A_i of the HLM acceptor, serve for this purpose. They preserve the non-terminal labels on the input side of the arcs in the composed transducer. This guarantees determinizability for the case that the HLM represents an ambiguous grammar. For the same reason, every arc returning from a branch describing possible pronunciations of a specific word carries the corresponding terminal label a_i on its input side. This keeps the composed automaton determinizable for the case of homophones, which occur if the same sequence of phonemes can be mapped to different terminal labels. Strictly speaking, the special opening label ϵ_{sub} is not necessary for disambiguation. Nevertheless it is treated just like a disambiguation label to prevent input ϵ labels, which would complicate subsequent composition operations.

[9]The semantic grammar for the airport information domain that has been used for the experiments conducted in the context of this thesis does not contain any recursive rule dependencies.

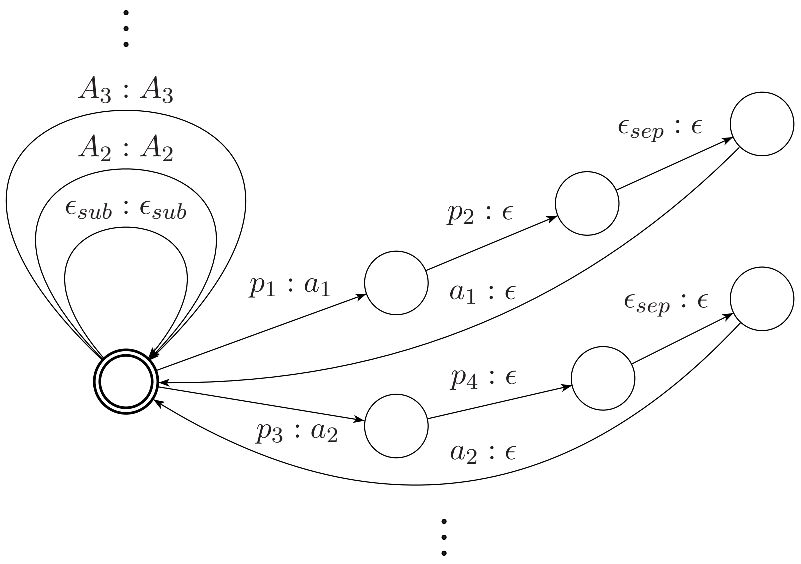

Figure 4.10: Layout of the phonetic lexicon transducer $\tilde{L}$ suited for the composition with the HLM acceptor $\tilde{G}$.

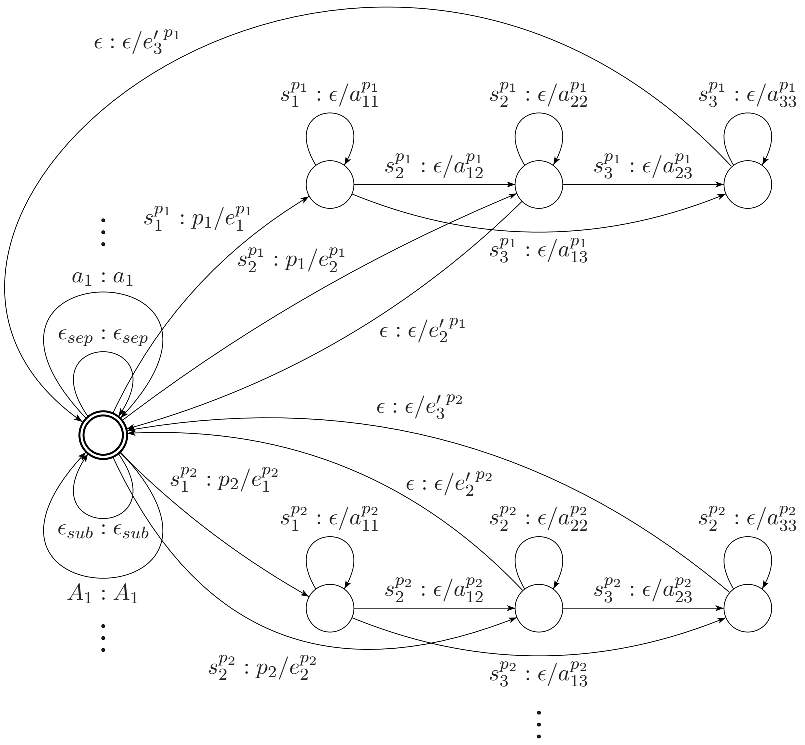
$\epsilon : \epsilon/e_3'^{p_1}$
$s_1^{p_1} : \epsilon/a_{11}^{p_1}$
$s_2^{p_1} : \epsilon/a_{22}^{p_1}$
$s_3^{p_1} : \epsilon/a_{33}^{p_1}$
$s_2^{p_1} : \epsilon/a_{12}^{p_1}$
$s_3^{p_1} : \epsilon/a_{23}^{p_1}$
$s_1^{p_1} : p_1/e_1^{p_1}$
$s_2^{p_1} : p_1/e_2^{p_1}$
$s_3^{p_1} : \epsilon/a_{13}^{p_1}$
$a_1 : a_1$
$\epsilon_{sep} : \epsilon_{sep}$
$\epsilon : \epsilon/e_2'^{p_1}$
$\epsilon : \epsilon/e_3'^{p_2}$
$s_1^{p_2} : p_2/e_1^{p_2}$
$\epsilon : \epsilon/e_2'^{p_2}$
$s_1^{p_2} : \epsilon/a_{11}^{p_2}$
$s_2^{p_2} : \epsilon/a_{22}^{p_2}$
$s_2^{p_2} : \epsilon/a_{33}^{p_2}$
$\epsilon_{sub} : \epsilon_{sub}$
$s_2^{p_2} : \epsilon/a_{12}^{p_2}$
$s_3^{p_2} : \epsilon/a_{23}^{p_2}$
$A_1 : A_1$
$s_2^{p_2} : p_2/e_2^{p_2}$
$s_3^{p_2} : \epsilon/a_{13}^{p_2}$

Figure 4.11: Layout of the acoustic model transducer $\tilde{H}$ suited for the composition with the result of $det(\tilde{L} \circ \tilde{G})$.

The special "seperator" label ϵ_{sep} is used to mark the end of words. Just like disambiguation labels, the seperator label has to remain on the input side during subsequent composition operations. This prevents its displacement with respect to time, which usually happens to output labels when applying the determinization operation (see figure 4.6). However, in contrast to disambiguation labels, which are removed after the final determinization operation, arcs carrying the seperator label are necessary to reconstruct the exact time alignment of the best matching path returned by the Viterbi decoder.

The lexicon transducer may also include complex pronunciation models, like the model for out-of-vocabulary words which has already been mentioned in section 3.3.3. The only prerequisite is that such a model can be represented by a weighted finite-state automaton, like for example an n-gram model predicting possible phoneme sequences.

4.3.3 Acoustic model transducer

The acoustic model consists of a collection of Hidden Markov Models that each represent a specific phoneme p_i. Hidden Markov Models, which have been introduced in section 2.2, are usually represented by automata in Moore format like the one of figure 2.1. In the usual case of continuous HMMs, a state s_i of such an automaton contains a specific probability density function $p(\mathbf{x}|s_i)$, which processes acoustic feature vectors $\mathbf{x}$.

Figure 4.11 shows the layout of the acoutic model transducer $\tilde{H}$ which maps from sequences of HMM states to phonemes. This transducer is suited for the composition with the transducer resulting from the determinization of the composition of the lexicon transducer $\tilde{L}$ and the HLM acceptor $\tilde{G}$. The result of $\tilde{H} \circ det(\tilde{L} \circ \tilde{G})$ maps from HMM state sequences to bracketed sequences of terminal and non-terminal labels. This represents the final search space over possible parse trees in $\tilde{G}$, which integrates the acoustic-phonetic and lexical knowledge sources $\tilde{H}$ and $\tilde{L}$.

Similar to the lexicon transducer where every word corresponds to a specific branch of the automaton that represents its pronunciation, the acoustic model transducer has branches which represent the Hidden Markov Models for each particular phoneme p_i. Such a branch provides the equivalent Mealy representation of the HMM, which means that the HMM state identifiers s_i now appear in form of input labels on the arcs of the automaton.

In figure 4.11 the HMM transition probabilities e_i, a_{ij}, and e'_i are included as weights on the corresponding arcs. Assuming the tropical semiring for the weight calculus during automata operations, the HMM transition probabilities have to be converted to arc weights on logarithmic scale. However, in the practical implementation, these arc weights are encoded as a part of the corresponding input labels.[10] The encoding of weights allows the hypothesis management of the Viterbi decoder to accumulate scores from acoustic model and scores from the HLM separately. This becomes necessary if the contribution of acoustic model and language model has to be reweighted, which happens for example during the estimation of confidence measures (see section 5.3.2).

[10]A transition weight is encoded into the input label by appending an identifier which is generated from the identifier of the tied transition matrix and the corresponding row and column indices.

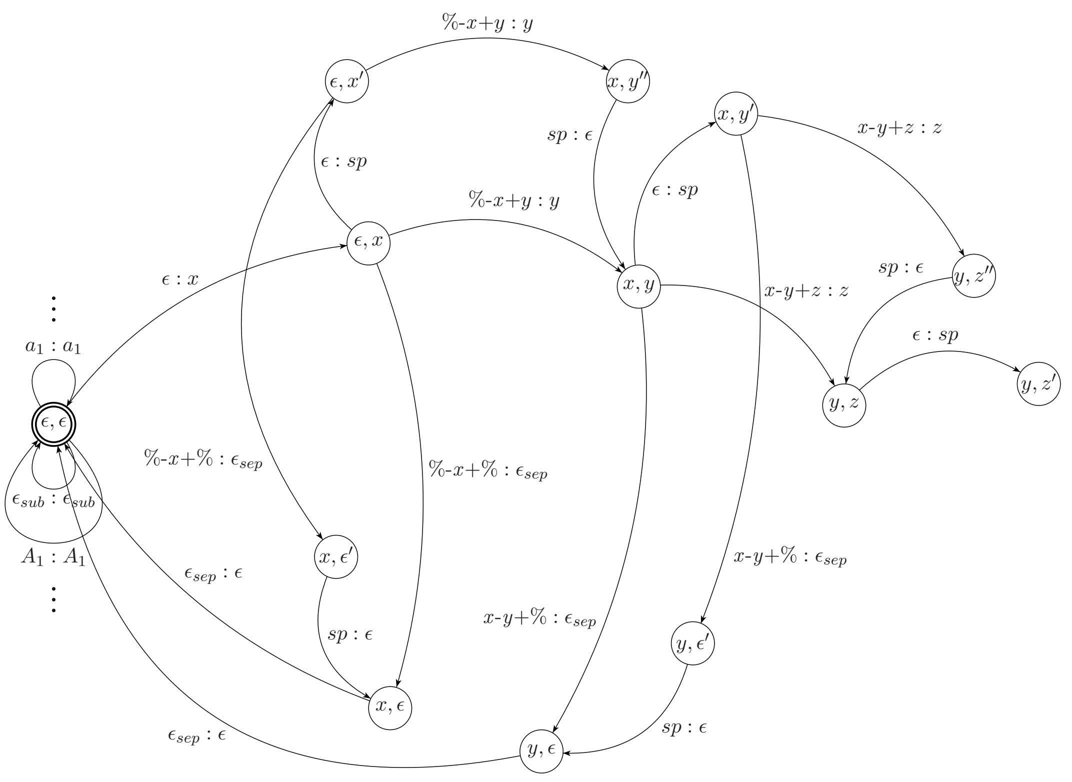

Figure 4.12: Snippet explaining the layout of the intra-word triphone transcuder $\tilde{C}$ suited for the composition with the result of $det(\tilde{L} \circ \tilde{G})$.

The encoding is resolved by a mapping from input label identifiers to pairs which contain the HMM state identifier and the corresponding arc weight. This mapping is initialized during the construction of the acoustic model transducer from the set of HMM parameter definitions for each phoneme.

Just like the lexicon transducer $\tilde{L}$, the acoustic model transducer $\tilde{H}$ contains self-loops at the initial state which carry disambiguation labels. During the composition with $det(\tilde{L} \circ \tilde{G})$, these self-loops preserve arcs carrying disambiguation labels which are necessary for the success of the subsequent determinization operation.

4.3.4 Triphone context-dependency transducer

As already mentioned in section 4.2.3, a special transducer can be applied in the case of context-dependent acoustic phoneme models, like triphones (see also section 2.6.2). This transducer is able to transform the outcome of $det(\tilde{L} \circ \tilde{G})$ with monophones on the input side to a transducer with triphones on the input side. This kind of transducers have also been introduced by Riley, Pereira and Mohri, who call them "context-dependency transducers" (see [RPM97]). For simplicity, in their publications the layout of the triphone context-dependency transducer is always presented for a phoneme set with only two phonemes. Figure 4.12 shows a snippet of the triphone context-dependency transducer, which explains its layout in greater detail, and also includes the special phoneme "*sp*" which models a short pause and is an example for a phoneme that is context-independent and must not be included in the context of other phonemes.

A triphone is denoted with "l-m+r", where l is the left, r the right context phoneme and m the phoneme itself. The transducer of figure 4.12 works for intra-word triphones. Thus, the first translated triphone of a word has an undefined left context "%-..." and the last translated triphone an undefined right context "...+%". Words with a single phoneme x are translated to the monophone "%-x+%". Each state corresponds to the left context and the middle phoneme of the triphones that occur on its outgoing edges. Thus, the translation from monophones to triphones is deferred by one phoneme (the monophone which is on the output side translates to the right context phoneme of the triphone label on the input side). This makes the handling of the context-free phoneme *sp* more complicated. A simple self loop that preserves the *sp*-phoneme on the input side does not work, because the *sp*-phoneme is displaced by one phoneme and occurs too early in the transducer which is the result of the composition operation. The correct processing of the *sp*-phoneme requires two auxiliary states for every state representing a specific combination of left context and middle phoneme. Figure 4.12 shows only a small part of all connections, just as much as are necessary to explain the principle of the combinatoric generation of the transducer, which is done by a threefoldly nested loop over the set of possible phonemes.

Actually, for the case of intra-word triphones, a context-dependency transducer is not absolutely necessary, because the triphones can be generated in a simpler way directly from the phonetic lexicon. However, the context-dependency transducer becomes essential, if the lexicon transducer contains a complex pronunciation model, like an out-of-vocabulary model. The context-dependency transducer is able to transform any finite-state model topology from monophones to triphones, which allows to train the out-of-vocabulary model on the basis of monophones and to augment the model with triphones during the runtime of the decoder.

Because the search space will be created on-demand during the decoding process, the set of occurring triphones is not known in advance, but is necessary for constructing the acoustic model transducer (see last section). Thus, the set of possible triphones has to be determined by preanalyzing the lexicon transducer $\tilde{L}$.

As stated in [RPM97], the context-dependency transducer is particularly suited for automatically constructing the complex search space layout for the case of large vocabulary regognition with cross-word triphones, where the network topology has to be split up for a correct junction of left and right context phonemes at word boundaries ("fanout"). Actually, the transducer $\tilde{C}$ could be further extended to support also cross-word triphones. This requires the special handling of the *sp*-phoneme and of the word end marker ϵ_{sep} to ensure the correct placing of these labels at the actual word boundaries in the resulting transducer. Furthermore, the self loops carrying the disambiguation labels, which for the intra-word triphone transducer only have to be generated at the initial state, have to be copied to the inner states of the cross-word triphone transducer, which could be done on-demand.

However, facing the great effort and the questionable performance gain in the considered application domain, which only has a medium sized vocabulary, it has been decided to conduct no experiments with cross-word triphones in the context of this thesis (the use of cross-word triphones even requires a special training of the underlying Hidden Markov Models, see [Beu99]).

4.4 Viterbi decoding of best-matching parse tree

The layout of the HLM acceptor preserves the hierarchical relations between non-terminal and terminal labels by bracketing each subnet instance by an initial arc carrying the opening label ϵ_{sub} and final arcs carrying the closing label, which is the label of the subnet itself. Thus, paths through the automaton implicitly represent the set of possible parse trees following the context-free grammar that corresponds to the HLM. By integrating the acoustic-phonetic and lexical knowledge sources via WFST operations, the best-matching parse tree can efficiently be determined directly from the speech signal, using the Viterbi algorithm and the token passing principle (see section 2.5).

4.4.1 Token passing in on-demand created search space

The main procedure of the Viterbi decoder is shown in pseudo-code fragment 4.8. It operates on the WFST representation of the search space N (variable "searchFst"), which results from the cascade of automata operations that combines the finite-state automaton representations of all involved knowledge sources[11] (see [MPR02]):

$$N = \pi_\epsilon(det(\tilde{H} \circ det(\tilde{L} \circ \tilde{G}))) \tag{4.1}$$

The operation π_ϵ replaces all disambiguation symbols on the input side with an ϵ-label and removes arcs that have an ϵ-label both on the input and the output side. The resulting search space transducer N maps from possible sequences of HMM states to bracketed sequences of non-terminal and terminal grammar symbols which represent possible parse trees.

The token propagation starts with the insertion of the "mother" token at the initial state of the search space transducer. The main loop of the search process is repeated as long as new feature vectors can be provided by the acoustic preprocessing module that processes the recorded speech signal. For every time frame all tokens have to be propagated in two consecutive steps. First, all tokens are repeatedly propagated until they reside at states that have outgoing "emitting" arcs, which means that the input label of such arcs refers to a specific HMM state. Then, all tokens have to be propagated a single time over these emitting arcs which includes the evaluation of the HMM state probability density functions with the current feature vector and the accumulation of the resulting acoustic score in every propagated token.

After the last feature vector, the repeated token propagation is carried out another time to assure that a token of the last time frame reaches the final state of the search space transducer, which has been discovered during the search process.[12]

[11]Formula 4.1 may also include the transducer for context-dependent phoneme models $\tilde{C}$ (see section 4.3.4)

[12]A search error because of a too tightly configured pruning threshold has happened if the final state has not been found or does not contain a token.

Pseudo-code fragment 4.8 Main procedure of time-synchronous Viterbi decoding.

```
procedure VITERBI(searchFst, prunDist, lmFactor)

    stateTokens ← new ARRAY()                        ▷ Manages token recombination.
    activeStates ← new QUEUE()          ▷ For repeated token propagation in the
current time frame.
    emittingStates ← new LIST()  ▷ For time-synchronous propagation of tokens
that pass arcs referring to HMM states.

    stateId ← searchFst.INITIALSTATEID()                ▷ Insert “mother” token at
    stateTokens[stateId] ← new TOKEN(0, undef, undef)               ▷ initial state.
    activeStates.ENQUEUE(stateId)
    finalStateId ← undef
    prunThres ← $\infty$
                ▷ While new feature vectors can be extracted from the speech signal...
    while CALCNEXTFEATUREVECTOR() do
                        ▷ Do repeated token propagation for the current time frame.
        PROPAGATEACTIVE(searchFst, activeStates,
                stateTokens, emittingStates, prunThres, finalStateId, lmFactor)
                ▷ Do time-synchronous propagation of tokens passing “emitting” arcs.
        PROPAGATEEMITTING(searchFst, stateTokens,
                    activeStates, emittingStates, lmFactor)
                                ▷ Determine pruning threshold for next time frame.
        prunThres ← CALCPRUNTHRES(prunDist)
    end while
                                        ▷ Propagate last tokens towards final state.
    PROPAGATEACTIVE(searchFst, stateTokens,
            activeStates, emittingStates, prunThres, finalStateId, lmFactor)
                                  ▷ No search error (token present at final state)?
    if finalsStateId $\neq$ undef and stateTokens[finalStateId] $\neq$ undef then
                        ▷ Reveal parse tree from sequence of recorded output labels.
        BACKTRACK(stateTokens[finalStateId])
    end if
end procedure
```

Starting from the token at the final state, the best-matching parse tree is revealed by backtracking the output label sequence, which has been recorded during the propagation of tokens.

The pseudo-code fragment 4.9 shows the subroutine PROPAGATEACTIVE() which executes the repeated propagtion of tokens until states with emitting arcs are reached. The recombination of tokens is managed by an array of tokens that dynamically grows with the state identifiers that result from the on-demand creation of the search space transducer (variable “stateTokens”). The propagation of a token over a specific automaton arc and its recombination at the destination state is done by the subroutine RECOMBINETOKEN() which is shown in the pseudo-code fragment 4.10. A token, which represents a specific alignment hypothesis that resides at a specific

Pseudo-code fragment 4.9 Does repeated token propagation until all tokens are waiting at states with “emitting” arcs.

```
procedure PROPAGATEACTIVE(searchFst, stateTokens,
        activeStates, emittingStates, prunThres, finalState)
                        ▷ While there are active states for the current time frame...
  while not activeStates.ISEMPTY() do
    stateId ← activeStates.DEQUEUE()
    token ← stateTokens[stateId]                    ▷ Gets token at active state.
    if token = undef then                         ▷ Token already propagated?
      continue                            ▷ Yes, continue with next active state
    end if
    if token.SCORE() < prunThres then                          ▷ Token pruned?
      token.RECORD()                           ▷ No, record token if necessary.
      isEmitting ← false                  ▷ Assuming state without emitting arcs.
      state ← searchFst.GETSTATE(stateId)         ▷ On-demand construction!
      for each arc of state do
        if not arc.ISEMITTING() then         ▷ Arc referring to HMM state?
      ▷ No, propagate token to destination state and perform recombination.
          RECOMBINETOKEN(stateTokens,
                          activeStates, arc, token, lmFactor)
        else
          isEmitting ← true              ▷ Indicate a state with emitting arcs.
        end if
      end for                       ▷ State with emitting arc or final state?
      if isEmitting = true or state.ISFINAL() then
        emittingStates.INSERT(stateId)      ▷ Yes, add state to list of states
waiting for the emission of the current feature vector.
        if state.ISFINAL() then               ▷ Final state has been found?
          finalStateId ← stateId                              ▷ Yes, save it.
        end if
      else stateTokens[stateId] ← undef            ▷ Clears propagated token.
      end if
    else stateTokens[stateId] ← undef                 ▷ Clears pruned token.
    end if
  end while
end procedure
```

automaton state, stores the accumulated score, the last passed automaton arc and a reference to the backtracking token. In the actual decoder implementation, the accumulated score is stored separately for acoustic and language model, which allows a later rescoring. Furthermore, the token saves the current time frame, which allows to reconstruct the temporal segmentation.

Before a token is propagated, a call to the method RECORD() checks if the token has entered the state over an arc that contains an output label that refers to the opening label ϵ_{sub} or to a grammar symbol, respectively contains the input label ϵ_{sep}, which marks a word boundary. In this case the token creates a copy of itself

Pseudo-code fragment 4.10 Propagates token over specific automaton arc and recombines it at the destination state. Returns score of recombined token.

```
function RECOMBINETOKEN(stateTokens, activeStates, arc, token, lmFactor)

                                              ▷ Add language model score from HLM.
    newScore ← token.SCORE() + arc.WEIGHT() * lmFactor
    if arc.INPUT() $\neq \epsilon$ then                 ▷ Arc encodes any HMM parameters?
        if arc.ISEMITTING() then                    ▷ Arc referring to HMM state?
            newScore ← newScore + GETHMMEMISSION(arc.INPUT)
                    ▷ Yes, add score for hmm emission density encoded in input label.
        end if
        newScore ← newScore + GETHMMTRANSITION(arc.INPUT)
                        ▷ Add score for hmm transition weight encoded in input label.
    end if

    destId ← arc.TARGET()                                   ▷ Get destination state.
    activeStates.ENQUEUE(destId)   ▷ Add destination state to active state queue.

    if stateTokens[destId] = undef then          ▷ Any token at destination state?
                                         ▷ No, create new token at destination state.
        stateTokens[destId] ← new TOKEN(newScore, arc, token.BACKTRACK())
    else if newScore < stateTokens[destId].SCORE() then          ▷ Recombination?
                                         ▷ Yes, replace token at destination state.
        stateTokens[destId] ← new TOKEN(newScore, arc, token.BACKTRACK())
    end if

    return stateTokens[destId].SCORE()        ▷ Return score of destination token.
end function
```

and refers to this copy by adjusting the reference to the backtracking token.

When a token is propagated to a destination state, this state is automatically inserted into the queue of active states (variable “activeStates”), which has to be made empty before the time-synchronous propagation via emitting arcs can take place. The selected queue discipline is not critical, because tokens will recombine at the latest in states where tokens wait for the time-synchronous propagation step.[13]

The time-synchronous propagation of tokens via emitting arcs is done by the subroutine PROPAGATEEMITTING(), which is shown in the pseudo-code fragment 4.11. Before their propagation, all tokens are moved from the array managing the recombination into a separate list (variable “emittingTokens”). This prevents the invalid recombination of tokens that correspond to different time frames.

[13] A partial topological order of the automaton states could prevent late recombination and may speed up the process of token propagation, but is difficult to determine because of the on-demand construction of the search space transducer.

Pseudo-code fragment 4.11 Time-synchronously propagates all tokens that wait at states with “emitting” arcs.

```
procedure propagateEmitting(searchFst,
        stateTokens, activeStates, emittingStates)

                        ▷ Copy tokens from recombination locations into a list.
  emittingTokens ← new List()
  for each stateId of emittingStates do
    token ← stateTokens[stateId]
    if token $\neq$ undef then                    ▷ Token already copied?
      emittingTokens.insert(token)               ▷ No, save reference in list.
      stateTokens[stateId] ← undef         ▷ Clear location of recombination.
    end if
  end for
  emittingStates.clear()

  for each token of emittingTokens do             ▷ Propagate all tokens in list.
    stateId ← token.getArc().target()             ▷ Get state from token's arc.
    state ← searchFst.getState(stateId)
    for each arc of state do
      if arc.isEmitting() then                  ▷ Arc referring to HMM state?
        score ← recombineToken(stateTokens,
                        activeStates, arc, token, lmFactor)
        updateBestScore(score)  ▷ For calculation of pruning threshold.
      end if
    end for
  end for
end procedure
```

4.4.2 Estimated rank pruning of improbable tokens

After the propagation over an emitting arc, the score of the resulting token is used to update the best score of the current time frame. The best score is needed to set the pruning threshold for the next time frame, which is used in the subroutine propagateActive() to eliminate improbable tokens, which are not propagated further if their score is below the pruning threshold. This allows a significant speed-up of the decoding process.

In the pseudo-code, the pruning threshold is simply determined by subtracting a fixed pruning distance (variable “prunDist”) from the best score of the last time frame. By the aid of the pruning distance, the trade-off between decoding speed and recognition performance can be adjusted. Because hypotheses are only considered inside a specific “beam” with respect to the best one, this simple method is called “beam pruning” (see [Low76]).

The disadvantage of the simple beam pruning method is the fluctuating processing load during the token passing search. To achieve a stable processing load, which is independent of the processed utterance, the number of tokens which have to be propagated in every time frame has to be bounded explicitly. Thus, the pruning distance has to be adjusted dynamically for every time step such that a fixed number of tokens will survive. A well-known method to achieve this is called “histogram pruning”; it determines the distribution of tokens over the range of score values in form of a histogram, which is then used to calculate the pruning distance that includes the selected number of tokens (see [STN94]).

However, the determination of the histogram costs computing time and memory space. This overhead can be saved by approximating the score distribution by a specific model. Therefore, the actual one-stage decoder implementation uses an approach, which assumes an exponential dependency between the number of tokens $N_i(t)$ and their distance t to the best score of the current time frame i (“estimated rank pruning”, see [JKO01]):

$$N_i(t) \approx a_i e^{b_i \cdot t} \tag{4.2}$$

The parameters a_i and b_i can be determined by counting the actual number of tokens for two settings t' and t''. These two values are simply set to the pruning distance estimated for the last time frame, $t' = t_{i-1}$, and to some fixed displacement in direction to the best score, which is $t'' = (1 - \delta)t_{i-1}$.[14] Then, the estimation for the pruning distance t_i for the current time frame is calculated by inserting the selected target for the number of tokens N_{opt} into the exponential distribution 4.2. This estimation is bounded by a given value that defines a minimum pruning distance and which is also used to initialize t' for the first time frame.

Because the assumption of an exponential score distribution is only an approximation, the actual number of tokens $N_i(t_i)$ scatters around the target value N_{opt}. Anyway, this pruning technique achieves an acceptable stability of the processing load, regarding the very little overhead which is necessary in comparison to other approaches.

4.5 Performance comparison of decoder implementations

In order to compare the performance of the decoder implementations for dynamic and static search space organization, the trade-off between decoding speed and achieved recognition accuracy is considered. The decoding speed is measured by the average of the real-time factor, which is the ratio of the decoding time and the actual length of a decoded utterance.[15] The recognition accuracy is specified by the socalled “tree node accuracy” which gives the fraction of correctly identified parse tree nodes which are relevant during the extraction of the slot-value pairs. A detailed description of this evaluation measure and of the experimental setup for the airport information domain used for testing will be given later on (see sections 6.1.2 and 6.2).

Figure 4.13 shows the performance comparison between static and dynamic search space organization for two different modeling setups. The modeling setup which achieves higher accuracy values, but also affords more decoding time, includes

[14]The selection of δ is uncritical; a value of 0.2 has been used, as suggested in [JKO01].

[15]All presented real-time factors were determined on a standard desktop system (2,8GHz PIV).

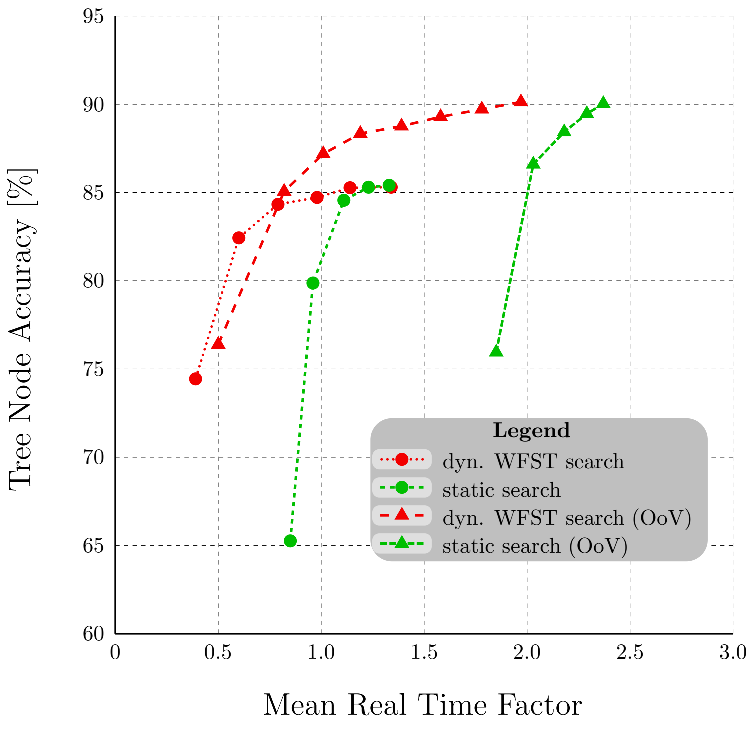

Figure 4.13: Performance comparison of decoder implementations that use the static and the dynamic search space organization.

a lexical out-of-vocubulary model which reduces the misinterpretation of ungrammatical utterance parts (see section 6.3.2). The curves depicted in figure 4.13 result from the variation of the maximum number of active tokens N_{opt} which controls the operation of the estimated rank pruning method that has been presented in the last section.

The decoder implementation using the WFST-based on-demand construction of the search space clearly outperforms the former decoder implementation using the static search space organization (see section 4.1). Obviously, the overhead of the on-demand construction costs much less processing time than the amount which is saved by the global search space optimization via the determinization operation. On the other hand, the results concerning the decoder implementation with the static search space organization show a minimum processing load that cannot be influenced by the pruning configuration, which inhibits the adjustment of real-time operation for the case of the modeling setup using the out-of-vocabulary model.

Thus, the goal to provide an implementation of the one-stage decoder that can be configured to operate under real-time conditions, even when using computationally expensive pronunciation models for out-of-vocabulary words, has been achieved by the aid of the powerful WFST calculus.

Chapter 5

Grammatical alternatives and semantic confidences

The generation of alternatives in form of word lattices or N-best lists is usually applied to make the sequential coupling of speech recognition and semantic interpretation more robust in comparison to the simple interface that only relies on the best-matching word sequence. The tight-coupling approach presented in the last chapter avoids the use of intermediately generated alternatives and directly determines the best-matching parse tree from the speech signal. Nevertheless, possible alternatives are of great interest, because the decoding process provides no certainty about the actual correctness of the resulting best-matching parse tree. The problem to measure the reliability of the content which is extracted from a user utterance is an important issue in the context of spoken dialog systems, because this provides a way to avoid interpretation errors and misguided dialog steps.

This chapter presents the compact representation of alternative parse trees by a hierarchy of lattices and explains its generation from the backtracking information which is stored during the one-stage decoding process. The lattice hierarchy representation provides the basis for an algorithm that allows the uniform estimation of semantic confidences for every particular node of a decoded parse tree. The presented approach is an extension of a well-known method to estimate confidences in word-based recognition systems by the calculation of posterior probabilities on word lattices. The chapter is concluded with a section that explains the translation of the estimated tree node confidences into confidences for the extracted slot-value pairs and how these confidences can be exploited by the dialog management of a spoken dialog system.

5.1 Word lattices

A word lattice serves for the compact representation of the part of the search space that was explored during the recognition of a specific utterance. Expressed more precisely, a word lattice captures the explored part of the search space in a form which is unrolled with respect to time. Therefore, a word lattice can be represented by an acyclic graph, which, in addition to the best-matching word sequence, contains other less probable alternatives. In a word lattice, the occurrence of a specific word is represented by an edge between two nodes that carry starting and ending time. The label of the word can be associated with either the edge or the node, which corresponds to the equivalent "Mealy" and "Moore" representation of automata (see end of section 3.1.2). In the context of this thesis, lattices are generated in the "Moore" representation, which means that every node corresponds to the end of a specific word. Anyway, an edge carries the score differences that resulted from the application of acoustic and language model given the corresponding word. The separation of acoustic and language model scores allows to "rescore" a word lattice, for example with a different language model.

Unfortunately, the term "word lattice" is not uniformly used in the technical literature about this subject. Some authors state that a "word lattice" only describes a table of unconnected word hypotheses, which had been the output of early speech recognition systems, and thus prefer the term "word graph". Nevertheless, in many publications both terms are used synonymously for the acyclic graph representation.[1] In addition to this inconsistency in terms, there is no common opinion about the actual algorithm that generates a word lattice. Several approaches to lattice generation have been published (e.g. [ON93, NA94]). However, because such an algorithm is always an inherent part of a particular speech recognizer and greatly depends on the underlying architecture, there is no standardized algorithm.

In the context of this thesis, the approach of "n-best" token passing has been used for the generation of lattices (see [YRT89]). In every time step of the token passing search, this approach records the n-best tokens that recombine at each word node in the integrated search network. Instead of a simple recombination, which discards a defeated token, the n-best approach ranks competing tokens due to their accumulated scores and chains them together in a linked list, just like the linkage to their predecessors with respect to time. The lattice is generated during the backtracking phase by tracing the recorded tokens in both linkage directions in a depth-first manner. Each created lattice edge represents a connection between two consecutively recorded nodes in the integrated search network. The acyclic graph structure of the lattice emerges from connecting tracked paths with existing lattice nodes, that have been created earlier in the ongoing lattice generation process. The resulting size of the lattice is affected by the maximum number of recorded tokens n and the tightness of the pruning configuration. As indicated in [YRT89], the lattices generated by this method do not necessarily include the global N-best sentences, but contain useful alternatives in addition to the best-matching sentence.

5.2 Flat lattice and lattice hierarchy representation

When applying the n-best token paradigm to the one-stage decoding algorithm for the weighted transition network hierarchy (WTNH, see section 3.3.2), the resulting lattices capture the imposed context-free grammatical constraints like in the example shown in figure 5.1. This snippet could be part of the lattice which is backtracked for an utterance like "*Der Flug nach Hamburg, zehn Uhr dreissig; wie ist die Flugnummer?*" (The flight to Hamburg, ten thirty; what is the flight code?).

[1] For example, in the widely used speech recognition toolkit HTK, the file format for the output of word graphs (and the input for language models) is called "standard lattice format".

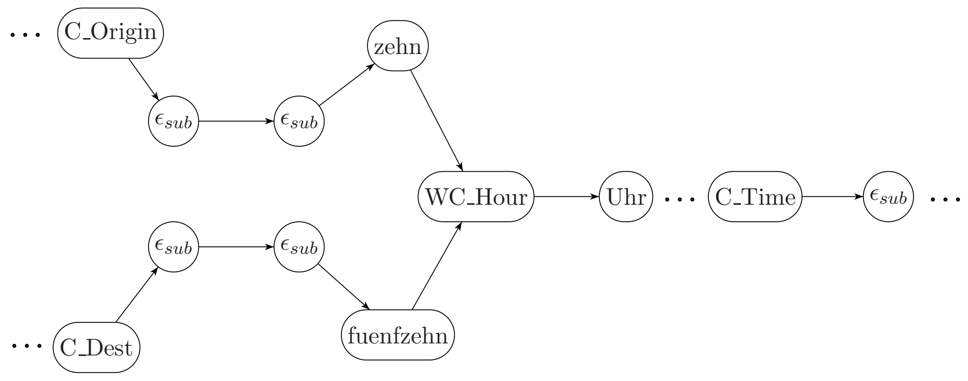

Figure 5.1: Snippet of exemplary flat lattice, that captures grammatical constraints by pairs of opening (ϵ_{sub}) and closing lattice nodes (e.g. “WC_Hour”).

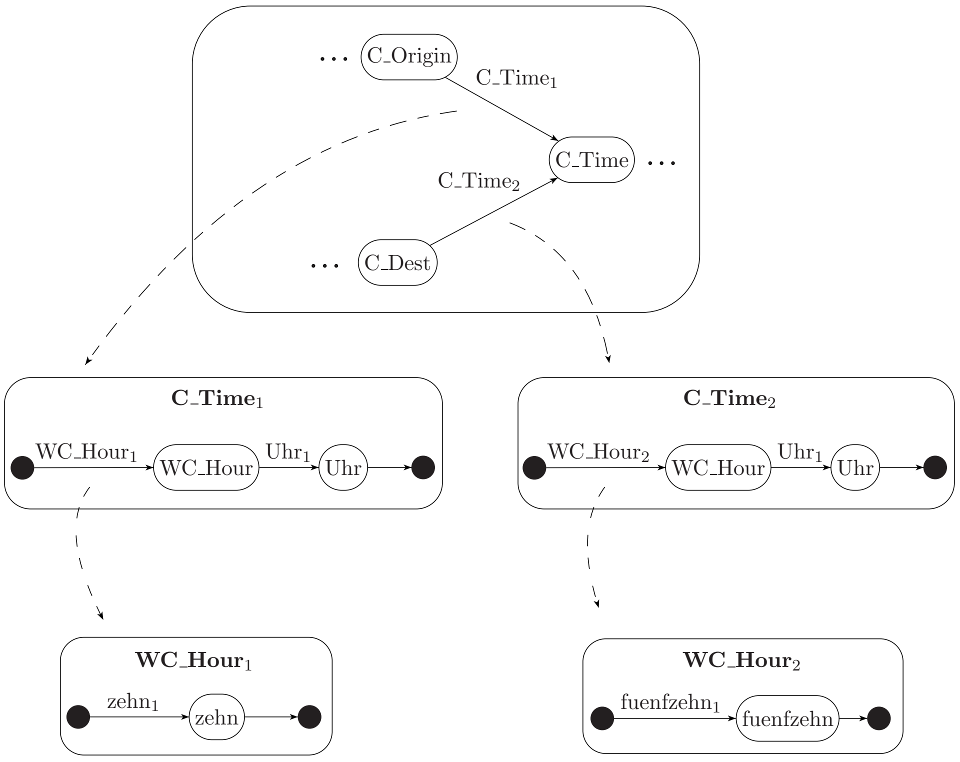

Figure 5.2: Snippet of lattice hierarchy that corresponds to flat lattice example.

Lattice nodes that carry the opening label ϵ_{sub} and a corresponding closing label ("C_..." or "WC_...") mark the start and the end of a specific occurrence of a particular non-terminal symbol of the corresponding context-free grammar. Due to the lack of space, the start nodes which correspond to nodes that carry terminal symbols (words) are not included in figure 5.1.

This representation captures grammatical constraints only implicitly, and therefore is called "flat" lattice. Possible alternatives on the level of a particular grammar rule can only be revealed by traversing the subordinate parts of the lattice which are located in-between consecutive nodes on that specific level.

Figure 5.1 shows two important features of the flat lattice representation:

- End nodes that mark the occurrence of a particular non-terminal symbol may correspond to several start nodes and vice versa. In the figure, the non-terminal symbol "C_Time" occurs with a single end node but two possible start nodes.
- Paths between different pairs of start and end nodes of a specific occurrence of a non-terminal symbol may intersect. In the figure, this is the case for the terminal symbol "Uhr", which lies on both paths that correspond to the non-terminal symbol "C_Time".

These properties motivate the following definition of a hierarchy of lattices, which allows an explicit representation of alternatives on each grammatical level:

- Every pair of connected start and end nodes defines a sub-lattice instance of the corresponding non-terminal symbol.
- Sub-lattice instances are referenced on corresponding edges inside their parent lattice instances.

All lattice instances consist of nodes that mark the end of a specific non-terminal or terminal symbol, and edges that each carry an instance index that identifies the corresponding sub-lattice instance. Edges that correspond to terminal words refer to trivial sub-lattice instances, which only contain a single word. Due to the limited drawing area, trivial word instances are not included in the graphical representation of lattice hierarchies. By definition, a sub-lattice instance always has a single entry and a single exit node, that correspond to the start and end node inside the flat lattice. Figure 5.2 shows the snippet of the lattice hierarchy which can be constructed on the flat lattice of figure 5.1. According to the structure of the flat lattice example, the lattice hierarchy contains two sub-lattice instances of the non-terminal symbol "C_Time" which share a single instance of the terminal symbol "Uhr".

The lattice hierarchy provides an explicit structural representation of the flat lattice, that allows to access alternatives on the level of a specific grammar rule without having to consider subordinate alternatives, which are encapsulated inside the sub-lattice instances. The following subsections explain the generation of the flat lattice from the backtracking information recorded during the n-best token-passing search, as well as the construction of the corresponding lattice hierarchy and the generation of ranked parse tree alternatives.

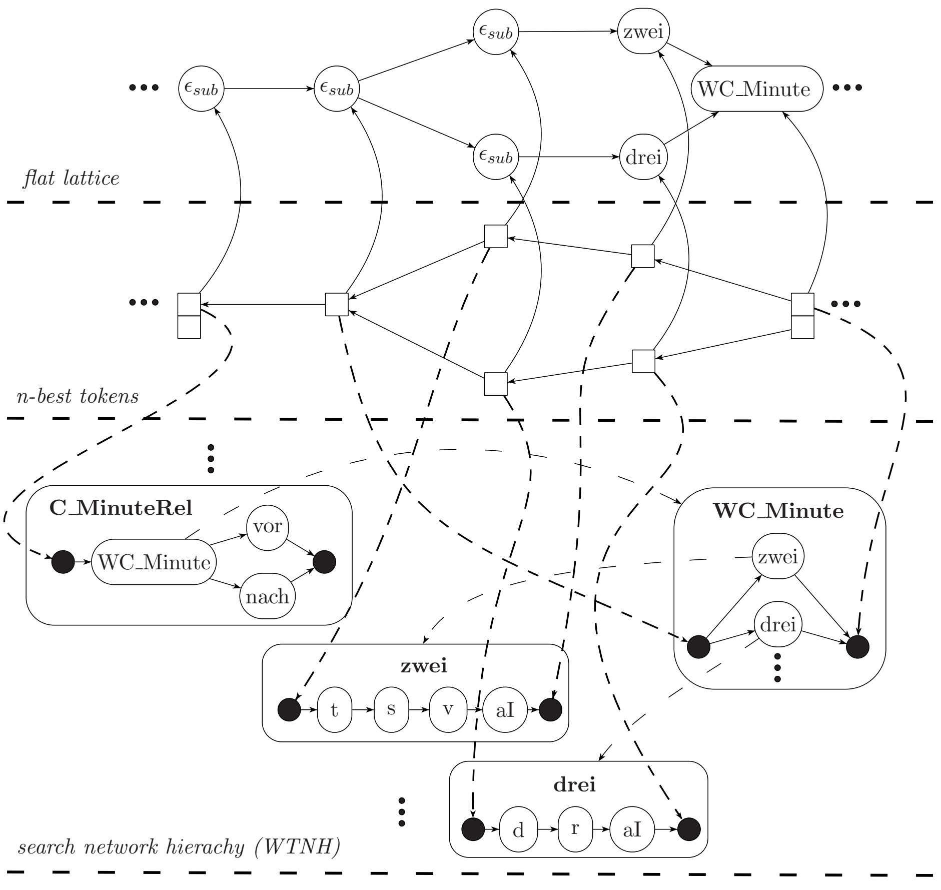

Figure 5.3: Example demonstrating the generation of the flat lattice for the static search space organization.

5.2.1 *N*-best token-passing for flat lattice generation

The generation of the flat lattice requires different implementations for the two architectures of the one-stage decoder which have been presented in the last chapter. In the case of the static search space organization, where the search network hierarchy is explicitly preserved, the generation of the flat lattice can be realized quite straightforwardly by evaluating the backtracking information recorded during the n-best token-passing search. The token passing principle inside the search network hierarchy has been illustrated by figure 4.1 (see p. 40), which serves as well to explain the n-best token passing: at each of the recombination locations, which are attached to the entry and the exit node of every subnet, the n-best incoming tokens are recorded for every time step of the processed speech signal. Because the segmentation is sufficient on word-level, tokens don't have to be recorded on the lowest subnet level that includes the acoustic models. Recording the n-best tokens at a specific recombination location actually means to link them together and to store the corresponding subnet node in a copy of the best token, which then is propagated further.

After the n-best token-passing search, the flat lattice is generated by tracing the

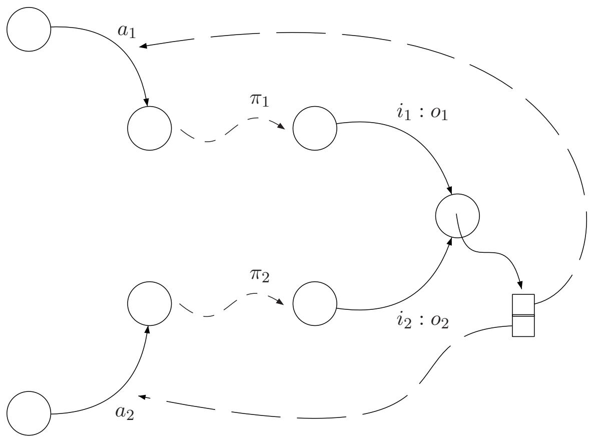

Figure 5.4: Illustration of the recombination of tokens inside the search space transducer that is constructed by on-demand WFST operations.

recorded tokens starting from the token recorded in the last time step at the exit node of the top-level network in the search network hierarchy. Because the tokens are recorded at the entry and at the exit node of each passed subnet, the revealed lattice has the required layout of the flat lattice that captures the grammatical constraints by marking the start and the end of a specific non-terminal or terminal symbol with corresponding lattice nodes. The lattice construction is carried out in a similar way like in the case of word lattices. Figure 5.3 illustrates the operation of the algorithm that constructs the flat lattice from the recorded n-best tokens, which are symbolized by the linked squares in the middle of the figure.[2] The first token of each n-best list refers to an entry or an exit node in the search network hierarchy, which is depicted in the lower part of the figure. The upper part of the figure shows the flat lattice, which is generated by a depth-first processing of the linked n-best lists. The arcs pointing from the n-best lists into the flat lattice stand for the generation of each particular lattice node.

For the decoder architecture which uses the dynamic search space construction by the on-demand concatenation of WFST operations (see section 4.4), the organization of the n-best token-passing and the following construction of the flat lattice becomes more difficult to implement. The reason for this is the fact that the search space transducer, which is the outcome of many complex finite-state automata operations, has no predictable layout. The determinization operation may move output labels to unpredictable locations inside the search space transducer which inhibits a straightforward handling of the recombination and the recording of the n-best tokens at locations which are known in advance, like in the case of the static search space organization.

Figure 5.4 illustrates the situation where two tokens recombine at a specific state of the search space transducer. For a correct handling of the recombination, it has to be decided whether an incoming token represents a new hypothesis with respect to the tokens, which already made their way to that particular state. In the case

[2]In figure 5.3, there are at most two tokens inside each n-best list.

of a new hypothesis, the token has to be inserted in the n-best list; otherwise, it has to be recombined with the token in the n-best list that corresponds to the same hypothesis. The first criterion for the recombination of tokens is whether the output labels o_1 and o_2 on the incoming automaton arcs are the same. If this is the case, the second criterion for recombination is that the automaton arcs a_1 and a_2, which have been recorded in the backtracking tokens, are also equal (π_1 and π_2 represent path sections over emitting arcs that only refer to HMM parameters and thus have not been recorded). If both conditions hold, the tokens have to be recombined, because they descend from the same state and don't represent different hypotheses.

Tokens are only recorded if they pass a non-empty output label or a word boundary which is marked by the special input label ϵ_{sep}. Thus, there is no guaranty that the n-best tokens are recorded at states where several paths of different hypotheses converge. In figure 5.4, this happens when o_1 and o_2 are the empty ϵ-label and the tokens have passed different arcs a_1 and a_2. Unfortunately, there is no straightforward way to identify these particular states in the progressively constructed search space transducer. Therefore, in contrast to the case of the static search space organization, not only the best token, but the whole n-best list is propagated to adjacent states, which assures that alternatives don't get lost and do reach a state where their recording takes place. The merging of n-best lists is organized in two steps: first, every token in the n-best list of the source state is propagated via the connecting arc into the n-best list of the destination state. If a matching token is found in the destination list, both tokens are recombined. If no matching token is found, the new token is appended to the destination list. After all source tokens have been processed, the resulting destination list is sorted due to the accumulated scores of the contained tokens. Finally, the length of the list is limited to the preset value of n by deleting all surplus tokens.

Beside the n-best token-passing, also the generation of the flat lattice itself becomes more complicated for the WFST decoding approach. The main difference with respect to the flat lattice generation for the static search space organization is that the search space transducer has "Mealy"-format (labels at arcs), which means that the lattice generation process has to convert the backtracking information into the "Moore"-format of the flat lattice (labels at nodes). Thus, during the generation of the flat lattice, the processing of a backtracked n-best list of tokens may include the creation and the connection of more than a single lattice node. On the other hand, if the recorded arcs of several tokens carry the same output label, only a single lattice node has to be created. This is the case in the example depicted in figure 5.5, where the two tokens of the n-best list, which corresponds to the rightmost state of the depicted part of the search space transducer, point to arcs that carry the same label "WC_Minute".

Another difficulty arises from the fact that, during the construction of the search space transducer, output labels referring to terminal word symbols may be shifted to any location inside the word, such that the correct temporal segmentation, as well as the corresponding acoustic score difference cannot be revealed correctly from subsequently recorded output labels. Therefore, word boundaries are explicitly marked by arcs that carry the special input label ϵ_{sep}, which are infiltrated into the search space transducer via the lexicon transducer (see section 4.3.2). In the resulting linkage of the recorded tokens, the information about the temporal segmentation

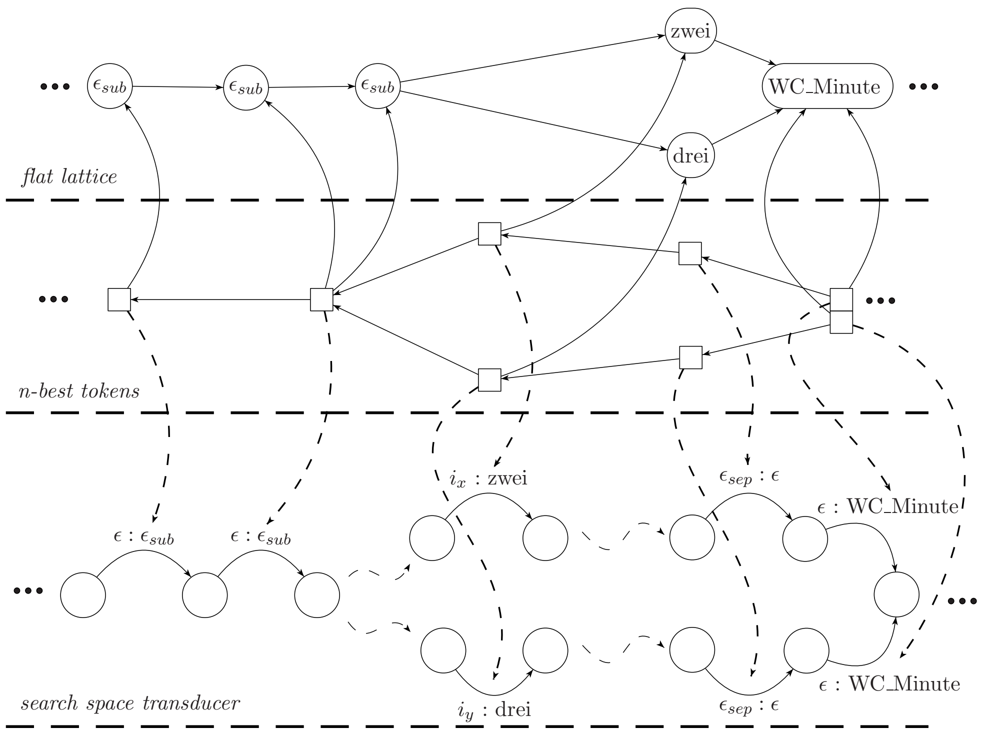

Figure 5.5: Example demonstrating the generation of the flat lattice for the dynamic search space organization via the on-demand composition of the search space transducer.

and the acoustic score difference appears "in between" the recorded labels of the corresponding words. For example, in figure 5.5 the lattice nodes for the words "zwei" and "drei" have to be created with references to the time step that has been recorded in the preceding tokens that refer to the arcs that mark the word boundary with the label ϵ_{sep}. The output labels "zwei" and "drei" can appear at any position inside the word, which is indicated by the undetermined input labels i_x and i_y that refer to particular HMM states and depend on how far the output labels are shifted towards the end of the word by the successive determinization operations that have been carried out for the on-demand construction of the search space transducer. The same consideration applies for the acoustic score differences of both words, which have to be calculated by the aid of the recorded word boundary as well. Generally, the construction algorithm can rely on the fact that a recorded word label is always preceded by a recorded word boundary and that this word starts at another word boundary or at a recorded start or end of a non-terminal symbol.

5.2.2 Construction of lattice hierarchy on flat lattice

The construction of the lattice hierarchy requires the creation of a sub-lattice instance for every pair of corresponding start and end nodes of particular grammar symbols, which can be found in the flat lattice. For a better comprehension of the operation of the construction algorithm, a complete example for a flat lattice and

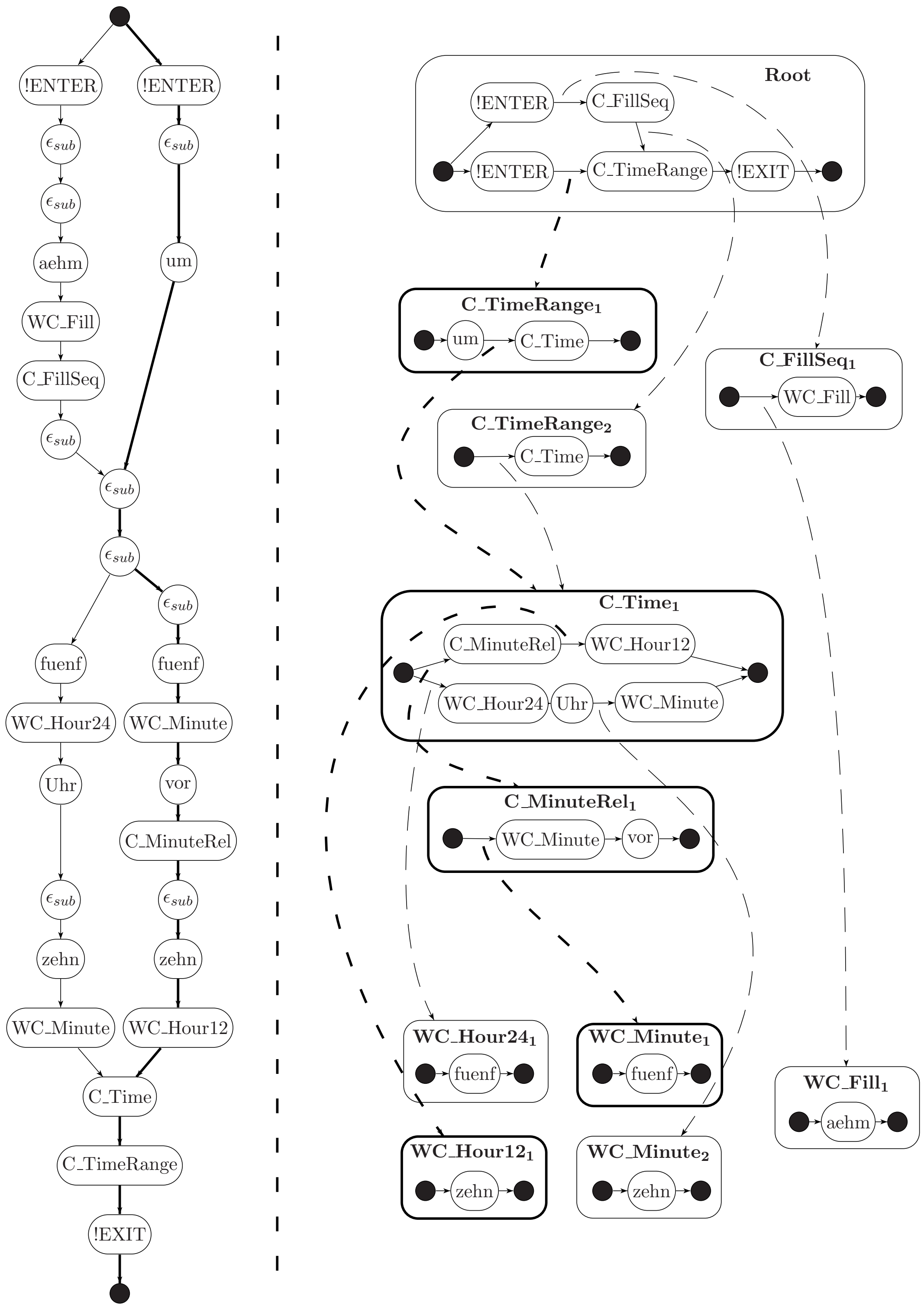

Figure 5.6: Example which shows the construction of the lattice hierarchy on the flat lattice.

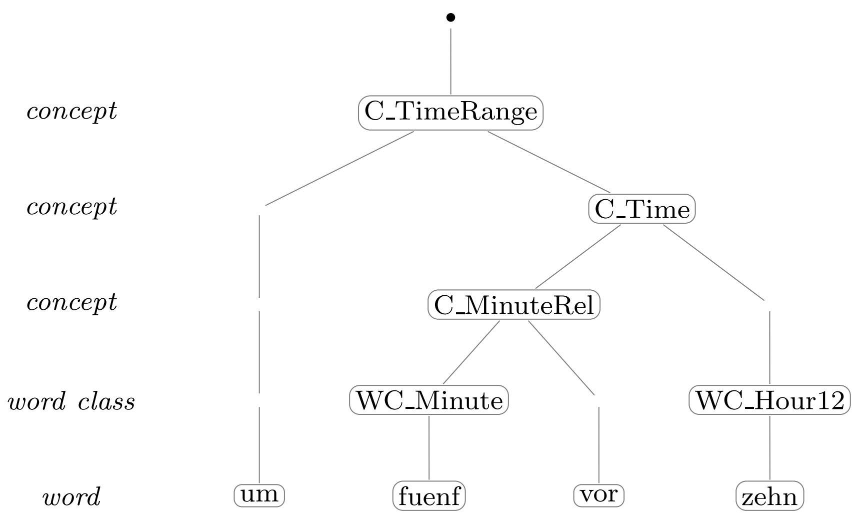

Figure 5.7: Best parse tree which corresponds to the sequence of visited sub-lattice instances when walking along the best path in figure 5.6.

the corresponding lattice hierarchy is given in figure 5.6. This example includes several utterances, which are phonetically similar in German language and express two different times of day: "*aehm fuenf Uhr zehn*" (uhm ten past five), "*um fuenf Uhr zehn*" (at ten past five), "*aehm fuenf vor zehn*" (uhm five to ten) and "*um fuenf vor zehn*" (at five to ten).[3]

The left side of figure 5.6 depicts the flat lattice which is generated during the backtracking phase of the n-best token-passing. On the right side of the figure, one can see the corresponding lattice hierarchy, which is revealed by processing the flat lattice nodes in depth-first order starting from the last node of the flat lattice. The recursive construction algorithm processes a flat lattice node in the following manner, depending on whether it is an end node or a start node:

- **End node.** If the current flat lattice node corresponds to the end of a non-terminal symbol, a new recursion on a lower sub-lattice level is initiated by the creation of a new temporary sub-lattice.
- **Start node (label ϵ_{sub}).** If the current flat lattice node corresponds to the start of a non-terminal symbol, the currently tracked path of the temporary sub-lattice is completed with the creation of an entry node. The recursive depth-first processing of the flat lattice is interrupted, which allows to reveal the rest of the temporary sub-lattice, including other possible entry nodes.

If all entry nodes of a temporary sub-lattice have been discovered, the construction of the lattice hierarchy can go on. A new sub-lattice instance has to be created for every entry node of the temporary sub-lattice. Each sub-lattice instance is generated by copying all nodes and edges which are found "between" the particular entry

[3]The word "aehm" is a German filler word which often occurs in conversational speech and which is explicitly represented by the grammar rules "WC_Fill" and "C_FillSeq", which model a sequence of filler words.

node and the single exit node of the temporary sub-lattice. At the same time, the temporary sub-lattice on the next higher level is updated by adding an edge for every created sub-lattice instance. Each edge refers to the corresponding sub-lattice instance by concatenating the name of its destination node and the allocated instance index (e.g. “WC_Minute$_1$ and “WC_Minute$_2$”). After the construction of the sub-lattice instances the depth-first recursion is continued from the flat lattice nodes that correspond to the entry nodes of the temporary sub-lattice, which can be discarded.

For example, in figure 5.6, after processing the node “!EXIT”, which is the terminal symbol for the piece of silence at the end of the utterance, the construction algorithm will create temporary sub-lattices for the encountered end nodes “C_TimeRange”[4], “C_Time” and “WC_Hour12”. The first sub-lattice instance which is created is “WC_Hour12$_1$”; the following are “WC_Minute$_1$”, “C_MinuteRel$_1$, “WC_Minute$_2$, “WC_Hour24$_1$, “C_Time$_1$”, etc. The temporary sub-lattice for the non-terminal symbol “C_TimeRange” has two different entry nodes, whose corresponding flat lattice nodes are connected with the nodes “C_FillSeq” and “!ENTER”, respectively. Thus, two sub-lattice instances for “C_TimeRange” are created which share the same sub-lattice instance of “C_Time”.

In order to prevent the creation of redundant paths inside sub-lattices it is important to remember the flat lattice nodes which already have been visited during the construction of the current temporary sub-lattice. If an encountered node is visited a second time, the currently tracked path is connected to the existing node of the temporary sub-lattice without continuing the depth-first recursion on the flat lattice. Just as well, each created sub-lattice instance has to be remembered with the corresponding end node inside the flat lattice. This allows to reuse sub-lattice instances when creating temporary sub-lattices with the same non-terminal symbol, which however correspond to a different end node inside the flat lattice.

Actually, the flat lattice also contains start nodes for nodes that carry terminal symbols, which however are not depicted in the graphical representations due to the lack of space. Therefore, the construction algorithm handles flat lattice nodes in the same way, no matter if they correspond to non-terminal or terminal symbols.

The best-matching path which is determined by the token-passing search is also the best path inside the flat lattice. In figure 5.6, this best-matching path is emphasized by the bold print of the corresponding lattice edges. Due to the depth-first processing of the recorded n-best tokens (best token first), the best-matching path is simply the first path which is revealed during the construction of the flat lattice. The corresponding best parse tree can easily be determined by projecting the best path of the flat lattice into the lattice hierarchy: The depth-first order of the parse tree nodes is exactly the order in which the corresponding sub-lattice instances are traversed when walking along the best path of the flat lattice. Figure 5.7 shows the parse tree that results from the order in which the bold-printed sub-lattice instances in figure 5.6 are visited, when walking along the best path.

[4]The non-terminal “C_TimeRange” also refers to a single time of day, which corresponds to a time range with zero duration.

5.2.3 Determination of the N-best parse trees

For the intended application of the one-stage decoder inside a spoken dialog system, a decoded parse tree has to be evaluated with the underlying semantic grammar to extract the contained slot-value pairs (see figure 3.4, p. 35). The dialog management may provide the ability to process several alternatives of probable slot-value pair collections, for example in order to have a backup in the case of a misguided dialog course, or in order to ask the user to clarify which of the alternatives he actually wanted to communicate to the system.

The lattice hierarchy compactly represents alternatives for probable parse trees including the best-matching one. However, the slot-value pair extraction may be able to process only a single parse tree at the same time. On the other hand, due to the one-stage decoding approach, the slot-value pair extraction affords little computational cost, because the semantic grammar, which contains the slot creation commands, is evaluated along the decoded tree without doing a full parse. Therefore, in order to get a ranked list of probable slot-value pair collections that can be passed to the dialog management, simply the N-best parse trees are determined and fed consecutively into the slot-value pair extraction.

For this purpose, an algorithm determines the N-best paths inside the flat lattice. The cost of a path is determined by accumulating the acoustic and language model score differences which are attached to the passed lattice edges. Each discovered path in the flat lattice corresponds to a particular sequence of visited sub-lattice instances inside the lattice hierarchy, which defines the according parse tree. The algorithm which is used to determine the N-best paths is called "recursive enumeration algorithm" (REA) and has been published by Jeminéz and Marzal (see [JM99]). An interesting feature of the REA algorithm is its ability to determine the N-best paths progressively, without having to specify the number N. This solves the problem that not every determined parse tree leads to a unique collection of slot-value pairs, which happens because parse trees may differ in nodes which are not considered during the slot-value pair extraction. With the progressive construction of the next best parse tree, the dialog management has the ability to continue the processing of next best parse trees until the desired number of unique slot-value pair collections have been extracted or no more paths are found in the flat lattice.

5.3 Estimation of semantic confidences

As already mentioned in the introduction of this chapter, alternatives in the form of lattices can be exploited for the estimation of confidences. Confidences measure the reliability of specific parts of a decoded recognition result, usually on word basis. The basic idea behind the lattice-based approach is that the confidence for a particular word in the lattice can be defined as its posterior probability. The posterior probability is calculated by the so-called "forward-backward algorithm" which is carried out on the lattice and considers the score values which are attached to the lattice edges.

The estimation of word confidences via posterior probabilities is a well-known method which has been discussed in great detail by Wessel et. al (see [WSMN01]). This publication confirms that in comparison with other approaches to the estima-

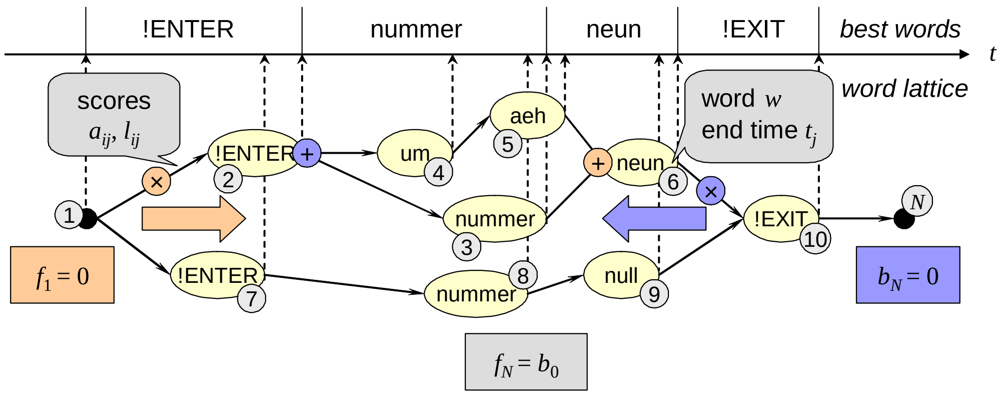

Figure 5.8: Example illustrating the calculation of word confidences by carrying out the forward-backward algorithm on the word lattice.

tion of confidences, like for example “acoustic stability” or “hypothesis density”, the posterior probability approach leads to the best results. In comparison to the identification of uninterpretable utterance parts via the use of explicit out-of-vocabulary models (see [TFLR05]), the posterior probability approach has the important advantage that no additional knowledge sources are needed for the confidence estimation.

The following subsections recapitulate the word-based posterior probability estimation and show how this method is extended to allow the estimation of semantic confidences which correspond to the nodes of a decoded parse tree. The tree node confidences are estimated by the aid of the lattice hierarchy representation which has been introduced in the last section. Furthermore, it will be shown how the estimated confidences are used to reduce the size of the lattice hierarchy, such that only the most probable alternatives are included. The final subsection explains the translation of the estimated tree node confidences into distinct confidences for the slot and the value of each extracted slot-value pair and discusses how these confidences can be exploited for the selection of the dialog strategy.

5.3.1 Word-based confidence measures

Figure 5.8 illustrates the estimation of word confidences on an exemplary word lattice. The confidence of a specific word w which is located between the lattice nodes i and j, can be expressed with the posterior probability $p\left([w; t_i, t_j] | \mathbf{x}_1^T\right)$, where t_i and t_j denote start and end time of the word w. The posterior probability is conditioned on the observed acoustic feature vector sequence $\mathbf{x}_1^T$ and can be expressed by the aid of forward and backward probability (see [WSMN01][5]):

$$p\left([w; t_i, t_j] \,|\mathbf{x}_1^T\right) = \frac{p_f(i)\, p\left(\mathbf{x}_{t_i}^{t_j} |\, [w; t_i, t_j]\right) p_b(j)}{\sum\limits_{\forall w_1^N} p\left(\mathbf{x}_1^T | w_1^N\right)} \tag{5.1}$$

[5]The difference to the definition in [WSMN01] is the assumption of a word lattice in “Moore”-format (word labels attached to nodes) and not in “Mealy”-format (word labels attached to edges).

$$p_f\left(i\right) = \sum_{\forall w_1^i \in F(i)} p\left(\mathbf{x}_1^{t_i} | w_1^i\right), \quad p_b\left(j\right) = \sum_{\forall w_j^N \in B(j)} p\left(\mathbf{x}_{t_j}^T | w_j^N\right) \tag{5.2}$$

$$\sum_{\forall w_1^N} p\left(\mathbf{x}_1^T | w_1^N\right) = p_f\left(N\right) = p_b\left(1\right) \tag{5.3}$$

The forward probability $p_f(i)$ is the sum of the probabilities of all paths w_1^i belonging to the set of paths $F(i)$ which start at the entry node and lead to the node i. In a similar manner, the backward probability $p_b(j)$ is the sum of the probabilities of all paths in the set $B(j)$, which includes the paths w_j^N that lead from node j to the exit node of the lattice. The denominator of equation 5.1 corresponds to the sum of the probabilities of all paths inside the flat lattice and can be determined either from the forward probability at the exit node, or from the backward probability at the entry node, which by symmetry have the same value. The probability $p(\mathbf{x}_{t_i}^{t_j} | \, [w; t_i, t_j])$ for the feature vectors $\mathbf{x}_{t_i}^{t_j}$ under the condition of the corresponding word $[w; t_i, t_j]$ is determined from the score differences for acoustic modeling a_{ij} and language modeling l_{ij}, which are attached to the lattice edge:

$$-\ln p\left(\mathbf{x}_{t_i}^{t_j} | \, [w; t_i, t_j]\right) = \alpha a_{ij} + \beta l_{ij} \tag{5.4}$$

By the aid of the scaling factors α and β it is possible to readjust the impact of acoustic and language model on the confidence estimation. Empirical tests showed that $\alpha = 1/s$ and $\beta = 1.0$ is an adequate setting which produces good results independently of the application domain (s is the language model factor which is used during the decoding process).

The forward, as well as the backward probability is determined by the recursive forward-backward algorithm that processes the score differences a_{ij} and l_{ij}. This happens by summing up the terms $\alpha a_{ij} + \beta l_{ij}$ along passed lattice edges (which is indicated in figure 5.8 by the symbol $\otimes$) and by carrying out the logarithmic sum over converging or diverging edges (indicated in figure 5.8 by the symbol $\oplus$).

For the calculation of the negative logarithm of the forward probabilities $f_i = -\ln p_f\left(i\right)$, the algorithm is:

$$f_1 = 0, \quad \forall i > 1: \quad f_i = \infty \tag{5.5}$$

$$\underset{top.\ order}{\forall} \quad i \in \{1, \ldots, N\}:$$

$$\forall k \in n\left(i\right): \quad \hat{f}_k = -\ln\left(e^{-f_k} + e^{-(f_i + \alpha a_{ik} + \beta l_{ik})}\right) \tag{5.6}$$

The recursive algorithm starts with the initialization (5.5) that sets the forward probability at the entry node of the lattice to one (zero on logarithmic scale) and the forward probabilities at all other nodes to zero (infinity on logarithmic scale, in practice a big number). Then, all lattice nodes have to be processed in topological order, which ensures that the score contributions of all predecessor nodes have been accumulated at the current node i, before this node is processed itself. For each of its successor nodes $n(i)$, the corresponding value f_k is updated via equation 5.6.

After processing the last lattice node, all f_i contain the negative logarithms of the corresponding forward probabilities.

The analogous algorithm to calculate the negative logarithm of the backward probabilities $b_j = -\ln p_b(j)$ is:

$$b_N = 0, \quad \forall j < N: \quad b_j = \infty \tag{5.7}$$

$$\underset{\textit{rev. top. order}}{\forall} \quad j \in \{N, \dots, 1\}:$$

$$\forall k \in p(j): \quad \hat{b}_k = -\ln\left(e^{-b_k} + e^{-\left(b_j + \alpha a_{kj} + \beta l_{kj}\right)}\right) \tag{5.8}$$

Here, the lattice nodes have to be processed in reverse topological order and the update formula 5.8 has to be carried out for all predecessor nodes $p(j)$ of the current node j.

The confidence $C([w; t_i, t_j])$ of a particular lattice edge $[w; t_i, t_j]$ is defined as the negative logarithm of the corresponding posterior probability, which has been defined in equation 5.1 and can now be expressed on logarithmic scale using the forward and backward scores f_i and b_j:

$$\begin{aligned} C([w; t_i, t_j]) &= -\ln p\left([w; t_i, t_j] \,|\mathbf{x}_1^T\right) \\ &= f_i + \alpha a_{ij} + \beta l_{ij} + b_j - f_N \end{aligned} \tag{5.9}$$

In figure 5.8, the best-matching word sequence is depicted above the word lattice. Usually, one is interested in the confidence for every word in the recognition result, which are simply the confidences of the corresponding lattice edges. The decision, whether a word should be accepted or rejected[6] is made by the comparison of the confidence with a specific threshold. By the variation of this threshold one can adjust the ratio of the number of errors committed by false acceptance and false rejection (see section 6.1.4).

Besides the simple definition (5.9), the authors of [WSMN01] suggest an extended confidence definition, which in addition to the best path considers all other paths in the word lattice that contain the particular word w for a time interval that intersects the time interval of the word w in the best path. This approach is motivated by the consideration that the confidence of a word is higher if it also occurs for similar time intervals in other paths than the best-matching one. For example, this is the case for the word "nummer" in figure 5.8.

Thus, the extended confidence definition takes the logarithmic sum over all confidences of words $[w; t_k, t_l]$ that intersect the time interval $\{t_i, \dots, t_j\}$ of the word w in the recognized word sequence:

$$C_{sec}([w; t_i, t_j]) = -\ln \sum_{\substack{\forall [w; t_k, t_l]: \\ \{t_k, \dots, t_l\} \cap \{t_i, \dots, t_j\} \neq \emptyset}} e^{-C([w; t_k, t_l])} \tag{5.10}$$

The extended confidence definition does not fulfill the original constraint of a posterior probability: C_{sec} does not necessarily meet the condition $C_{sec} \geq 0$ and thus may

[6] What is actually meant by "accepted" and "rejected" depends on the application of the confidences, e.g. the deletion of rejected words.

correspond to an invalid probability which is greater than one. In practice, however, the definition C_{sec} consistently leads to better results than the simple definition C, which has been confirmed empirically by the authors of [WSMN01], who carried out various experiments on different test corpora.

5.3.2 Estimation of parse tree node confidences

After the introduction of the word-based confidence estimation, this section explains the extension of the underlying posterior probability approach to support the estimation of confidences for each node of a decoded parse tree (see also [LFRT04]). The confidence of a parse tree node should quantify its reliability independently of its child nodes. For example, if a decoded parse tree contains the node "C_Time", the corresponding confidence should allow a decision about whether the user mentioned a time of day at all or not, no matter what particular time has been recognized. The reliability for this particular time of day is given by the confidences for the tree leafs that contain the corresponding words. Later on, it will be explained how the confidences for parse tree nodes are transformed into confidences for slots and values, that are extracted from a decoded parse tree (see section 5.3.4).

As stated before, each node of a decoded parse tree corresponds to a particular sub-lattice instance in the constructed lattice hierarchy (see end of section 5.2.2). Such a sub-lattice instance is denoted by $[I_L; t_i, t_j]$; L is the label of the sub-lattice instance, which is equal with the label of the corresponding tree node; t_i and t_j denote the time steps which are attached to the nodes i and j inside the flat lattice that correspond to the entry and the exit node of the sub-lattice instance. Equation 5.9, which gives the definition for the word confidence, can be modified to allow the calculation of the confidence of a sub-lattice instance that corresponds to a specific parse tree node:

$$C\left([I_L; t_i, t_j]\right) = f_i + f_{ij} + b_j - f_N \tag{5.11}$$

f_i and b_j represent the forward and the backward score, which are calculated for the start node i and the end node j of the sub-lattice instance inside the flat lattice. The symbol f_{ij} replaces the original term $\alpha a_{ij} + \beta l_{ij}$ in equation 5.9 and denotes the forward score of the part of the flat lattice that is located between the start and the end node of the sub-lattice instance. The calculation of f_{ij} is similar to the calculation of the original forward score (see recursion formulas 5.5 and 5.6), but only considers flat lattice nodes between the entry node i and the exit node j:

$$f_i = 0, \qquad \underset{i<p\leq j}{\forall} p: \quad f_p = \infty \tag{5.12}$$

$$\underset{top.\ order}{\forall} p \in \{i, \ldots, (j-1)\}:$$

$$\forall q \in n(p): \quad \hat{f}_q = -\ln\left(e^{-f_q} + e^{-(f_p + \alpha a_{pq} + \beta l_{pq})}\right) \tag{5.13}$$

After processing all involved lattice nodes, the value for f_{ij} is equal to the forward score f_j, which has been accumulated at the end node j. When determining the confidence for a leaf node of the parse tree that contains a particular word, f_{ij} is just the original term $\alpha a_{ij} + \beta l_{ij}$, because the corresponding sub-lattice instance refers to a single edge in the flat lattice.

The fact that word lattices may contain the same word with intersecting time intervals motivated the extended word-confidence definition C_{sec} (see equation 5.10). Similarly, the lattice hierarchy may contain several sub-lattice instances which have the same label and intersecting time intervals. Therefore, the confidence definition for a parse tree node that corresponds to the sub-lattice instance $[I_L; t_i, t_j]$ is extended to consider all other sub-lattice instances in the lattice hierarchy that have the same label L and intersect the time interval $\{t_i, \ldots, t_j\}$:

$$C_{sec}\left([I_L; t_i, t_j]\right) = -\ln \sum_{\substack{\forall\, [I_L; t_k, t_l] : \\ \{t_k, \ldots, t_l\} \cap \{t_i, \ldots, t_j\} \neq \emptyset}} a_{sec}\left(t_i, t_j, t_k, t_l\right) e^{-C([I_L; t_k, t_l])} \tag{5.14}$$

In order to achieve a better approximation of the logarithmic probability constraint $C_{sec} \geq 0$, equation 5.14 contains the intersection ratio a_{sec}, that scales posterior probabilities according to the degree of intersection of the time intervals of the current sub-lattice instance $[I_L; t_k, t_l]$ and the sub-lattice instance $[I_L; t_i, t_j]$, which corresponds to the node of the decoded parse tree, for which the confidence is estimated. Assuming intersecting time intervals, a_{sec} is calculated by

$$a_{sec}\left(t_i, t_j, t_k, t_l\right) = \frac{\min\left(t_j, t_l\right) - \max\left(t_i, t_k\right)}{\max\left(t_j - t_i, t_l - t_k\right)} \tag{5.15}$$

As expected from the results reported on word-basis, the extended confidence definition C_{sec} performs better than the simple definition C when evaluating the quality of the confidence estimation for parse tree nodes (see section 6.3.1).

5.3.3 Confidence-based pruning of flat lattice

A practical problem results from the fact that a backtracked flat lattice can get very large, depending on the parameter selection for the n-best token passing search (pruning settings and the value of n). Unfortunately, these settings don't allow a proper filtering of improbable path alternatives, such that it is necessary to generate a large flat lattice to ensure that the majority of probable path alternatives is included.

The computational cost for the construction of the lattice hierarchy on a large flat lattice may cause a noticeable delay in the reaction of the dialog system, which could be avoided by removing improbable paths from the flat lattice. A quite obvious approach to solve this problem is to prune the flat lattice by the aid of the estimated confidences. For word lattices, this strategy has been used successfully by the authors of [WSMN01]; they applied the lattice pruning approach presented in [SO99].

In the present work, the strategy of exploiting confidence values to delete improbable paths from the flat lattice has been realized in the following way: the criterion for pruning a flat lattice edge is the distance of its confidence with respect to the average confidence of the lattice edges that belong to the best path through the flat lattice. The pruning criterion is only evaluated for flat lattice edges that correspond to words and therefore consume a specific time interval $\{t_l, \ldots, t_k\}$. The condition for deleting a word edge $[w; t_l, t_k]$ from the flat lattice is:

$$C\left([w; t_k, t_l]\,|\mathbf{x}_1^T\right) \quad > \quad \frac{\sum\limits_{t=t_l}^{t_k} C_{best}\,[t]}{t_k - t_l} \quad + \quad C_{dist} \tag{5.16}$$

The confidence $C([w; t_k, t_l]\,|\mathbf{x}_1^T)$ is determined via the estimation formula 5.9. The array $C_{best}[t]$ contains the confidence values of the word edges that belong to the best path for each time step t of the acoustic signal preprocessing. For example, for a particular lattice edge $[w_{best}; t_i, t_j]$ on the best path, the values $\{C_{best}[t_i], \ldots, C_{best}[t_j - 1]\}$ are all set to the specific confidence value $C([w_{best}; t_i, t_j])$. Thus, the condition 5.16 for deleting a word edge refers to the distance to the average confidence of the best path, which is calculated over the corresponding time interval.

The pruning condition is tested for each word edge inside the flat lattice, which is encountered during a depth-first search in forward direction. During this depth-first search, a particular word edge is only passed if its confidence doesn't meet the pruning condition 5.16. A subsequent depth-first search in backward direction finally determines all co-accessible[7] lattice nodes and eliminates all "dead ends" in the pruned flat lattice.

An important feature of this approach is the fact that the best path in the flat lattice is always preserved, because it provides the reference confidence in the pruning condition 5.16. The maximum confidence distance C_{dist} is a free parameter, just like the pruning distance for the token-passing search, which has to be adjusted by empirical tests. During such tests, a drastic reduction of the average flat lattice sizes could be achieved without negatively affecting the overall quality of the confidence estimation. At the same time, the computational cost for the construction of the lattice hierarchy and the calculation of the tree node confidences can be kept on a low level, such that there is no noticeable delay in the reaction of the dialog system.

5.3.4 Slot- and Value confidences

The estimation method which has been presented in section 5.3.2 allows to determine a confidence for every node of a decoded parse tree. However, inside a spoken dialog system, the dialog management processes a collection of slot-value pairs which is extracted from a decoded parse tree. Thus, it is necessary to find a reasonable way to assign the estimated tree node confidences to the extracted slot-value pairs.

To tap the potential of the tree node confidences, it has been decided to define a separate confidence for the slot and the value for every extracted slot-value pair (see also [LTR+05]). For example, if the decoded parse tree contains a time of day, one can distinguish between the confidence that the user uttered a time of day at all (slot confidence), and the confidence for that particular time which has been extracted (value confidence). Please note, that this distinction is not possible by using simple word confidences, because a slot may have high confidence, although the confidences for the particular words that determine its value are low.

[7] A "co-accessible" node can be reached from the start node of the lattice when passing the lattice edges in forward direction, as well as from the end node when going in backward direction.

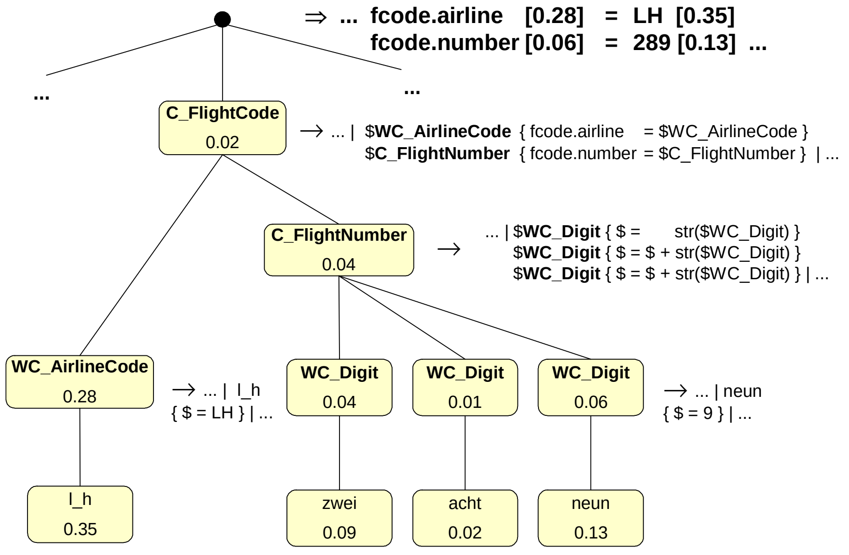
⇒ ... fcode.airline [0.28] = LH [0.35]
fcode.number [0.06] = 289 [0.13] ...
...
...
C_FlightCode 0.02
→ ... | $WC_AirlineCode { fcode.airline = $WC_AirlineCode }
$C_FlightNumber { fcode.number = $C_FlightNumber } | ...
C_FlightNumber 0.04
→ ... | $WC_Digit { $ = str($WC_Digit) }
$WC_Digit { $ = $ + str($WC_Digit) }
$WC_Digit { $ = $ + str($WC_Digit) } | ...
WC_AirlineCode 0.28
→ ... | l_h { $ = LH } | ...
WC_Digit 0.04
WC_Digit 0.01
WC_Digit 0.06
→ ... | neun { $ = 9 } | ...
l_h 0.35
zwei 0.09
acht 0.02
neun 0.13

Figure 5.9: Example showing the extraction of slot-value pairs and their corresponding confidences from a decoded parse tree that contains a flight code.

In order to translate tree node confidences into slot and value confidences, a rule-based technique is employed. It considers the way in which the slot extraction commands of the semantic grammar are applied during the post-processing of a decoded parse tree (see section 3.2.2). For the case that a slot is created and filled without referring to subordinate slots, the slot confidence is set to the confidence of the current tree node, in which the slot is created. The value confidence is calculated by taking the maximum over the confidences belonging to the sibling nodes in the scope of the semantic tag, that contains the slot creation command. In the other case, where subordinate slots are involved, the confidence of the newly created slot is set to the maximum over the confidence of the current tree node and the slot confidences of the subordinate slots. The value confidence of the new slot is set to the maximum of the value confidences of the subordinate slots.

The reason for choosing the maximum operator to combine several tree node confidences, is the pessimistic assumption that the worst confidence of the particular set of involved tree nodes determines the confidence of the corresponding slot or value.[8] Figure 5.9 shows the extracted slot and value confidences that result from the depicted parse tree when applying the rule-based approach as specified above. In the case that the user may have made a self-correction, such that a specific slot is extracted a second time, the new value replaces the old value only if the confidence of the new slot is below the confidence threshold used for discarding slots.

A simple policy for exploiting the slot and value confidences in the dialog management is the following: If the slot confidence is above a certain threshold, the slot is completely discarded. If the slot confidence holds, but the value confidence is above the threshold, the dialog management initiates a step to clarify the slot

[8]Because beeing defined as the negative logarithm of a posterior probability, a big value means a bad confidence.

content. If slot and value confidences are both accepted, the dialog management can safely rely on the slot-value pair when making decisions that determine the subsequent dialog steps.

In addition to the best-matching parse tree, the slot and value confidences can be determined for the N-best parse trees, too. However, an investigation on sophisticated dialog management techniques that, when processing a user utterance, are able to exploit several alternatives for possible slot-value pair collections by the aid of the contained semantic confidences is beyond the scope of this work. By all means, the semantic confidences presented in this work are particularly worthful in conjunction with approaches to statistically driven dialog management (see for example [YWS$^+$05]).

Chapter 6

Evaluation methods and experimental results

In order to test the developed one-stage speech interpretation system and the estimation of semantic confidences, several recognition experiments have been carried out on a corpus of user utterances that has been recorded during the simulation of a spoken dialog system providing an airport information service (NaDia research project, see section 1.3).

This chapter includes a detailed description of the properties of the airport information corpus and explains the experimental setup which has been used to produce the presented results. The experimental results are expressed by means of several evaluation measures that quantify the achieved interpretation performance.

Apart from verifying the gain in computational efficiency that is achieved by the one-stage decoder implementation using the finite-state transducer technique (see section 4.5), the conducted experiments have the main purpose to evaluate the quality of the semantic confidence estimation discussed in the last chapter.

6.1 Off-line evaluation methods

The evaluation methods used in the context of this thesis only consider the speech interpretation task without taking into account the other components of the spoken dialog system. The common principle of such off-line evaluation methods is a comparison of the recognition outcome with a reference transcription that provides the correct result. The comparison of "hypothesis" and "reference" reveals the committed errors, which are accumulated over the collection of test utterances. The result of the evaluation is a recognition rate which is determined from the observed error counts.

The considered evaluation methods differ in the kind of the transcription which is the subject of the comparison between hypothesis and reference. The simplest case is a word-based transcription, which however does not consider the variable information content of specific words. In order to get a better measure for the achieved interpretation performance, the evaluation is carried out on the basis of decoded parse trees and on the transcription of extracted slot-value pairs (see sections 6.1.2 and 6.1.3).

The obtained results for the information extraction performance allow to compare different system setups in order to verify improvements achieved by adjusting free system parameters or altering the knowledge sources involved in the speech interpretation task. However, it is questionable whether the off-line evaluation of the information extraction performance provides an adequate quality measure for speech interpretation with respect to its application inside the spoken dialog system, since the history of consecutive dialog turns cannot be considered. This would require a test environment that actually allows the interaction between user and dialog system. If conducted with real users, such tests are expensive to perform and do not allow extensive experiments for optimizing the system configuration, like they are possible when doing an automatic off-line evaluation. In order to solve this problem, there are approaches to simulate the user, such that the dialog system can be evaluated automatically (at least on the basis of utterance transcriptions, excluding the speech recognition task). User simulation techniques also allow to apply machine-learning approaches to dialog management (see for example [SGY05]). This complex subject is however beyond the scope of this work, which will only report results obtained by off-line evaluations.

6.1.1 Word-based evaluation

The traditional performance measure used in automatic speech recognition is the “word error rate”. This measure is based on the mapping with “minimum edit distance” that is determined for a recognized word sequence and its corresponding reference transcription. A specific mapping between two symbol sequences $S_1[i]$ and $S_2[j]$ is defined by a list of particular “edit operations” $i \mapsto j$, which are necessary to transform the first symbol sequence into the second one (each symbol denotes a specific word). Possible edit operations are “substitution”, “insertion” and “deletion”. A substitution can either be “correct” or “wrong”, depending on whether $S_1[i] = S_2[j]$ is true. An insertion or deletion means that either i or j denote the empty symbol ϵ. For the calculation of the edit distance, one has to define a cost function. The commonly used NIST standard defines the following costs: 4 for a wrong substitution, 3 for insertion and deletion and 0 for a correct substitution (see [FF93]). This cost definition prefers a substitution with respect to the equivalent succession of insertion and deletion. The edit distance of two symbol sequences are the accumulated costs for all involved edit operations. In order to find the minimum edit distance, one can apply the “dynamic programming” algorithm that also returns the corresponding sequence mapping (see [SK83]).

To determine the word error rate on a specific collection of test utterances, it is necessary to determine the sequence mapping with minimum edit distance for each test utterance and to accumulate the number of correct substitutions N_C, the number of wrong substitutions N_S, the number of insertions N_I and the number of deletions N_D. The word error rate is defined as:

$$WER = \frac{N_S + N_D + N_I}{N_C + N_S + N_D} \tag{6.1}$$

The denominator $N_C + N_S + N_D$ is equal to the number of words in the reference transcription. Because this definition counts insertions as errors, the word error rate is not bounded to one. The recognition performance can also be expressed with the equivalent recognition rate, called “word accuracy”:

$$\begin{aligned} Acc &= 1 - WER \\ &= \frac{N_C - N_I}{N_C + N_S + N_D} \end{aligned} \tag{6.2}$$

The word-based evaluation treats all words equally and therefore assumes that each word carries the same amount of information. This assumption is acceptable for applications where the recognized word sequence is more or less the final result, like in the case of an automatic dictation system. For the task of automatic speech interpretation, the assumption that each word of a recognized utterance contributes equally to the extracted information is definitely wrong. Furthermore, the word-based evaluation does not consider the grammatical structure that guides the semantic interpretation.

The consequence of these shortcomings is that the word-based evaluation does not provide a reliable measure for the speech interpretation performance: there is no guarantee that a system configuration which is optimized on the word error rate also achieves the best possible error rate with respect to the extracted information (see also [WAC03]). The following subsections present evaluation methods which take into account the structure of the decoded parse trees or directly consider the extracted information on the basis of slot-value pairs.

6.1.2 Tree-based evaluation

Because the result of the one-stage decoding process are parse trees, it stands to reason to extend the sequence-based evaluation method in a way that it allows the comparison of parse trees. It is actually possible to define the basic edit operations “substitution”, “deletion” and “insertion” for tree nodes and to provide a dynamic programming algorithm that determines the minimum “tree edit distance” between two trees (see [SZ97]). This “tree matching” algorithm has successfully been applied to determine a “tree node accuracy” that allows to measure the average structural concordance of a test set of decoded parse trees with their corresponding reference parse trees (see [TFLR03]).

The reference parse trees are automatically generated by parsing the orthographic transcription of the test utterances with the semantic grammar. This means that the evaluation is actually a comparison of the output of the one-stage decoder which processes the speech signal with the output of the robust parser which processes the manually transcribed utterance representation. Therefore, the evaluation does not reveal conceptual errors of the handcrafted semantic grammar. Consequently, a prerequisite for the evaluation is a consistent grammar design that allows to process the orthographic transcription of all test utterances correctly, such that every relevant piece of application-specific information can be extracted from the reference parse trees.

Figure 6.1 shows an example for the alignment of a decoded parse tree and its corresponding reference parse tree. This alignment is the outcome of the tree matching algorithm which finds the minimum tree edit distance. The determined alignment classifies each involved tree node either as correct or substituted (tree

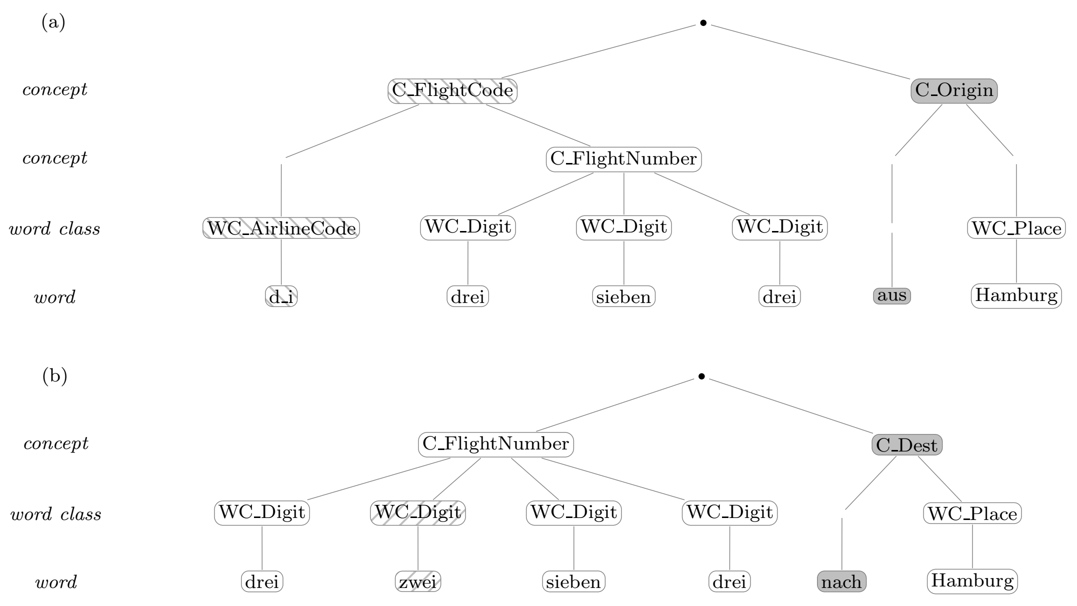

Figure 6.1: Example for the alignment of hypothesis (a) and corresponding reference parse tree (b) that identifies correct, substituted, inserted and deleted tree nodes.

nodes emphasized in gray color) or as inserted or deleted (tree nodes emphasized by hatch pattern, insertions in the hypothesis and deletions in the reference).

The tree matching is effected on all pairs of hypothesis and reference parse trees of the test set, such that the total number of correct, substituted, inserted and deleted tree nodes can be determined (N_C, N_S, N_I, and N_D). By inserting these counts in equation 6.2 one obtains the so-called tree node accuracy, which specifies the fraction of correctly identified tree nodes (less the incorrectly inserted ones) with respect to the total number of tree nodes in the collection of reference parse trees.

As indicated on the left hand side of figure 6.1, the nodes of a parse tree correspond to different hierarchy levels that result from the nested structure of the corresponding grammar rules. This allows to subdivide the tree nodes due to their hierarchy level and to carry out the evaluation for each partition of tree nodes separately. It has been decided to define three different hierarchy level types: the "word" level, the "word class" level and the "concept" level. The word level simply refers to the leaf nodes of the parse trees which always contain words (terminal symbols). The word class level includes all "pre-terminal" tree nodes, which contain only a single word. These nodes (and the corresponding grammar rules) are indicated by the prefix "WC_...". All higher-ranked tree nodes are assigned to the concept hierarchy level, which is indicated by the prefix "C_...".[1] The hierarchy level type is also considered during the tree matching: a substitution is only allowed if both tree nodes have the same level type.

The separate evaluation for each hierarchy level type allows a more detailed analysis of the obtained results. For example, the separate evaluation shows that the

[1]The prefixes have the additional purpose to prevent name clashes between non-terminal and terminal symbols.

proposed method for the estimation of tree node confidences works uniformly well, independent of the hierarchy level of a particular parse tree node (see section 6.3.1).

Parse trees may contain nodes that are not relevant during the extraction of the information that is further processed by the dialog management. For example, in the airport information domain, phrase parts like "Where...", "At which gate...", or "Please tell me the gate..." may all be handled by a specific grammar rule that fills the slot for the subject of the user request with the value "gate", independently of the exact wording. The evaluation, however, should consider only those tree nodes which are actually involved in the final information extraction. The consideration of irrelevant parse tree nodes can be prevented by manually classifying particular non-terminal symbols as "type relevant" or "value relevant". This classification scheme is used to select all relevant nodes of a given parse tree: Each type relevant node is preserved as well as all of its parent nodes, up to the root of the parse tree. In turn, each value relevant node additionally preserves all of its child nodes. The filtering is applied to every pair of hypothesis and reference tree, before the tree matching is carried out. For example, if the non-terminal symbols "C_Origin" and "C_Dest" are classified as type relevant and the non-terminal symbol "WC_Place" as value relevant and the filtering is carried out for figure 6.1, the word nodes "aus" (from) and "nach" (to) are deleted from hypothesis and reference tree. This is actually intended, because the filtering prevents that the error of confusing arrival and departure is counted twice.

6.1.3 Performance of information extraction

As explained in section 3.2.2, the semantic grammar provides embedded scripting commands that control the slot-value pair extraction. During parsing, those rules are applied to extract a specific collection of slot-value pairs for the resulting parse tree. If the parse tree is the output of the one-stage decoder, the slot-value pair extraction is done in a post-processing step that applies the semantic grammar along the decoded parse tree.

Consequently, the evaluation can be carried out on the basis of slot-value pairs. This actually means to compare the slot-value pair collections which are extracted from the decoded parse trees with the corresponding slot-value pair collections that are extracted by parsing the orthographic transcriptions of the test utterance set. When comparing the hypothesis and the reference slot-value pair collection of a particular utterance, a slot is counted as correct if it can be found in both hypothesis and reference and if the values of both slots are equal. If the values are not the same the slot is counted as substituted. An insertion is counted if the slot occurs only in the hypothesis, a deletion if it occurs only in the reference. A sequence aligning is not necessary, because the slots are processed independently of the order of their extraction.[2]

The comparison of hypothesis and reference slot-value pair collections on a test set provides the total number of correct (N_C), substituted (N_S), inserted (N_I) and deleted (N_D) slots. Common measures for the performance of information extraction

[2]A specific slot may occur only once in the slot-value pair collection extracted for a particular utterance, because for the case that a slot is extracted multiple times, the corresponding value is replaced in the order of occurrence.

are the "precision" (P) and "recall" rate (R), as well as the "F-Measure", which are all defined by means of the observed slot counts (see [MKSW99]):

$$P = \frac{N_C}{N_C + N_S + N_I} \tag{6.3}$$

$$R = \frac{N_C}{N_C + N_S + N_D} \tag{6.4}$$

$$F = \frac{2PR}{P + R} = \frac{N_C}{N_C + N_S + \frac{1}{2}(N_I + N_D)} \tag{6.5}$$

The F-Measure has been introduced to have a single performance measure and is defined as the harmonic mean of precision and recall rate. This definition, however, has been criticized by the authors of [MKSW99], because it attenuates the negative effects of insertions and deletions. They suggest a definition which is equivalent to the word error rate (see equation 6.1) which they call accordingly "slot error rate". In this work, results are uniformly given in form of recognition rates. Thus, the results presented at the end of this chapter will be expressed by the obtained "slot accuracy" (1 - "slot error rate", see equation 6.2).

A legitimate question is, whether the tree-based evaluation is still valuable when having the possibility to do the evaluation on the basis of the extracted information. A potential advantage of the tree-based evaluation is the better "resolution" due to the pieces of information, which are combined when doing the slot-value pair extraction. On the other hand, the argument that a slot should be counted as totally wrong, even when parts of the contained information are correct, is also valid because in many cases it is highly questionable whether the dialog system can really profit from extracted information that is only partially correct.

In the context of this discussion one already reaches the limits of what can be achieved by an off-line evaluation of the speech interpretation module, without considering the whole dialog system and the history of consecutive dialog turns (see beginning of section). For producing the results presented in this work, it has been decided to adjust the parameter configuration of the speech interpretation system on a separate "cross-validation" utterance set to achieve a maximum tree node accuracy for nodes which are relevant for information extraction. This parameter configuration is then applied to produce the presented results on the actual test utterance set. The obtained results have the main purpose to quantify the improvements which can be achieved by exploiting the estimated tree node confidences.

6.1.4 Evaluation of confidence estimation

In order to measure the quality of the estimated parse tree node confidences (see section 5.3.2), two different evaluation methods are used: the "confidence error rate" and the "receiver-operator characteristic". These measures are suggested in [WSMN01] for the evaluation of word confidences. Thanks to the tree matching algorithm, both measures can also be used for the evaluation of the confidences for parse tree nodes (see also [LFRT04]).

In order to decide whether a specific node of a decoded parse tree should be accepted or rejected, it is necessary to define a particular threshold ε. The tree node is classified as "accepted" if its confidence is below this threshold, otherwise it is "rejected". For the evaluation of the quality of the confidence estimation one has to count the errors which are committed when making this decision: if a tree node is classified as accepted, but the tree matching algorithm identifies this node either as "substituted" or "inserted", an error is counted for "false acceptance". Respectively, if the tree node is classified as rejected, but is actually "correct", an error is counted for "false rejection". In this way one can determine the total number of falsely accepted and falsely rejected tree nodes over all test utterances (N_{FA} and N_{FR}).

The confidence estimation can only identify errors which are caused by substitution or insertion. Nodes which are deleted with respect to the reference parse trees cannot be identified. Thus, the confidence error rate CER is defined as the ratio of the total number of errors committed by false acceptance and false rejection and the total number of tree nodes in the hypothesis:

$$CER = \frac{N_{FA} + N_{FR}}{N_C + N_S + N_I} \tag{6.6}$$

The confidence error rate depends on the setting of the classification threshold ε, which is adjusted on the test utterance set used for cross-validation. In order to measure the improvement which can be achieved by considering the confidence values, the confidence error rate is compared with the "decoder baseline". The baseline is the confidence error rate which results from classifying all tree nodes as "accepted", without taking into account any confidence value at all. Thus, the baseline CER_{BL} includes errors that result from the false acceptance of substituted and inserted tree nodes:

$$CER_{BL} = \frac{N_S + N_I}{N_C + N_S + N_I} \tag{6.7}$$

If the confidence estimation performs well, the confidence error rate drops below the decoder baseline, because the analysis of the confidences identifies more substitutions and insertions than it produces errors on correctly decoded parse tree nodes.

Another appropriate method to measure the quality of the confidence estimation is the receiver-operator characteristic, which is depicted by "ROC-curves". This diagram shows the false acceptance rate FAR and the false rejection rate FRR for a variable setting of the classification threshold ε. The false acceptance rate is the ratio of the number of false accepted and the number of substituted and inserted tree nodes. Respectively, the false rejection rate is the ratio of the number of false rejected and correctly decoded tree nodes:

$$FAR = \frac{N_{FA}}{N_S + N_I}, \quad FRR = \frac{N_{FR}}{N_C} \tag{6.8}$$

The ROC-curve shows the trade-off which is achieved between false acceptance and false rejection rate. For a well performing confidence estimation, the ROC-curve runs close to abscissa and ordinate of the diagram.

In section 5.3.4 it has been explained how to translate the estimated tree node confidences into confidences for the slot and the value of each slot-value pair that is extracted from a decoded parse tree. The benefit of the determined slot confidences can be evaluated by the gain of the information extraction performance (see equations 6.3 et. seqq.) which can be achieved by discarding slots that have bad confidence. This evaluation also allows a comparison with the use of an explicit out-of-vocabulary model, that directly recognizes utterance parts which cannot be interpreted with the semantic grammar (see section 6.3.2).

6.2 Airport information corpus

The application domain used in the context of this thesis is a spoken dialog system that offers an airport information service in German language. The usage scenario assumes that the system is part of the man machine interface of future automobiles, which besides the conventional haptic-visual interface allows the interaction via natural speech. When going to the airport, the driver can use the airport information system to request various information about arriving or departing flights: the exact arrival or departure time, the flight number, the airline, the origin or destination of a flight, the gate, the status (on time or delayed), the airplane type and the appropriate location to park his vehicle. In order to obtain this information, the user has to provide either the flight code or the rough arrival or departure time and the destination or origin of the flight. The dialog system allows a “mixed initiative” interaction, which means that the dialog starts with an open-ended system prompt (”How may I help you?”) and the user does not need to follow a predetermined dialog flow. In the case that the user has difficulties to provide the information necessary for his query, the dialog behaves cooperatively by asking the user to clarify his intent or to give particular information. The user may follow the suggested dialog flow, but can take the initiative at any time by changing the subject under discussion or by rejecting the last system prompt in order to correcting himself or the system which committed an interpretation error.

6.2.1 Data collection

The development of essential parts of a spoken dialog system, like the speech interpretation module, affords the collection of application-specific data. This data collection, however, requires a runnable version of the spoken dialog system. A common solution to this problem is to conduct so-called “Wizard-of-Oz” experiments (see [DJA93]). In such experiments a human “wizard” simulates the missing system modules. The test subjects are usually not aware of this simulation. They are told that they are interacting with a fully automatic system, which makes the test environment more realistic.

The Wizard-of-Oz experiment conducted for the airport information domain consisted of a number of tasks which had to be performed by the test subjects. Each task was characterized by a short description which was read to the test subjects by the supervisor responsible for conducting the experiment (e.g. “Your friend is on a machine which arrives from Nuremberg between five and six o’clock in the evening. Find out the exact arrival time!”). During listening to the task description, the test subjects had to memorize the necessary information for their next interaction with the airport information system. In order to cover the range of functionality of the airport information application, each test subject had to complete about sixteen different tasks. The deployed Wizard-of-Oz system already included a working version of the dialog management module. Therefore, the major task of the supervisor was

	lab	car
# test subjects	38	17
# user utterances	3414	1323
# words	10191	7642

Table 6.1: Amount of data collected for the airport information domain in laboratory and car environments.

to simulate the speech interpretation module (speech recognition and extraction of slot-value pairs). The system response was generated automatically from the manually extracted information and checked by the supervisor before playing it to the test subject. During the manual information extraction, the supervisor deliberately committed interpretation errors to provoke user utterances for rejecting or correcting the system response.

The airport information corpus has been collected in several test series in two different environments (see table 6.1). About two thirds of the corpus has been recorded under "laboratory conditions", where the test subjects sat in front of a desktop computer system that ran an artificial driving task for simulating the distraction caused by driving the car and keeping track of the traffic situation. The rest of the corpus was collected under realistic conditions inside a driving car which was steered by the test subjects during a ride through the city traffic of Munich. In both environments, the voice of the test subjects was recorded by a close-talking microphone, in order to minimize the negative influence of background noise.

6.2.2 Experimental setup

The collected airport information corpus has been split into three parts. The training utterance set ("train"), which comprises about two thirds of the whole corpus is reserved for developing the handcrafted semantic grammar and for estimating the stochastic weights of the corresponding hierarchical language model. The remaining part of the corpus serves for testing the one-stage decoder and the information extraction with the constructed hierarchical language model. The test corpus is again split into two equal sized parts. The first part, called "cross-validation" utterance set ("xval"), is used for finding the optimum settings for all free parameters involved in the one-stage decoding process, like for example the language model scaling factor or the threshold setting when exploiting the estimated confidences. The second part is the actual test utterance set ("eval"), which serves for producing the results presented in this work using the parameter settings adjusted on the cross-validation utterance set. Table 6.2 shows the exact partitioning of the airport information corpus into the three sub corpora.

The handcrafted semantic grammar comprises 570 different words (terminal symbols) and 82 different rules (non-terminal symbols). The embedded slot creation commands allow the extraction of 35 different slots. The parse tree annotation is generated by applying the semantic grammar on the orthographic corpus annotation via the use of a robust parser. Table 6.2 shows the total number of annotated tree nodes and the corresponding average per utterance for each of the three hierarchy

	train	eval	xval
# test subjects	43	6	6
# user utterances	3752	448	537
# word nodes	12892	2541	2400
# words nodes per utt.	3.4	5.7	4.5
# word class nodes	5300	1112	1082
# word class nodes per utt.	1.4	2.5	2.0
# concept nodes	10976	1836	1856
# concept nodes per utt.	2.9	4.1	3.5
# slots	5957	985	1072
# slots per utt.	1.6	2.2	2.0
# different word nodes	730	295	266
# different word class nodes	14	10	11
# different concept nodes	68	48	47
# different slots	35	24	25
unmatched word rate	7.7%	7.6%	5.2%

Table 6.2: Partitioning of airport information corpus and statistics of the parse tree annotation which is generated with the aid of the handcrafted semantic grammar.

level types introduced in section 6.1.2 ("word", "word class" and "concept"), as well as the corresponding figures for the extracted slots. The "unmatched word rate" specifies the fraction of words which during the robust parsing could not be assigned to any grammatical rule and thus are classified as "out-of-vocabulary".[3]

The reader may have noticed that, in the average, the training corpus contains shorter utterances than the test sub corpora and that the test sub corpora do not cover the complete semantic grammar. The reason for this circumstance is that the semantic grammar contains rules for domain-independent user commands that are necessary to integrate the flight information application into the greater context of a demonstration system (for example commands for switching the domain, getting help, etc.). Because the corresponding user utterances are very short and simple, they have been excluded from the evaluation.

The parse tree annotation of the training corpus is used to estimate the stochastic weights of the hierarchical language model (HLM), which is created from the semantic grammar by converting it into a transition network hierarchy (see section 3.3.3). The top-level network of the HLM used for the experiments contains a bi-gram over all main rules of the semantic grammar.

Apart from the semantic-syntactic model (semantic grammar and corresponding HLM), the experimental setup also includes the acoustic-phonetic modeling. The phonetic lexicon has been created manually by the aid of comprehensive phonetic lexicon resources (e.g. Phonolex, see [pho04]). The acoustic modeling is done

[3]The detection of these words in the training corpus can be exploited for automatically determining the locations where to integrate an out-of-vocabulary model inside the hierarchical language model (see section 3.3.3). When occurring inside the sub corpora which are used for testing, those words are simply deleted before carrying out the evaluation.

Acc_{word}	Acc_{tree}	Acc^{rel}_{tree}	F-Measure	Acc_{slot}
78.6%	79.9%	85.4%	90.3%	83.9%

Table 6.3: Results of the experiment with the baseline speech interpretation system; determined on the basis of words (Acc_{word}), parse tree nodes (Acc_{tree}), relevant parse tree nodes (Acc^{rel}_{tree}) and slot-value pairs (F-Measure, Acc_{slot}).

by speaker-independent continuous-density Hidden Markov Models for intra-word triphones. For the creation of speaker-independent acoustic models, the airport information corpus is too small. Therefore, it has been decided to use a larger data collection from the German Verbmobil project for the acoustic model training. The application scenario of this data collection are dialogs about the scheduling of appointments between two human dialog partners (the major goal of the Verbmobil project was automatic translation, see [Wah00]).

The used version of the Verbmobil data collection includes about 11000 spontaneous utterances in German language of about 600 different speakers, which is a sufficient size for creating speaker-independent Hidden Markov Models. The training is effected with the popular HTK-Toolkit by following the strategy proposed in the provided documentation (see [YEK+02]). It begins by the creation of an initial set of context-independent HMMs for 50 different German phonemes, including a silence model for short and long pauses and a model for non-speech sounds. The HMMs have “Bakis” topology (see figure 2.1, p. 14) and include one, three or four emitting states depending on the phoneme (a single emitting state for the short pause, four states for diphthongs and tree states for all other phonemes). The HMM probability densities are modeled by Gaussian distributions with diagonal covariance matrices.

The initial context-independent HMMs are extended to intra-word triphone HMMs by using the common parameter tying approach (see section 2.6.2). The quality of the HMMs is further improved by applying the suggested method of “mixture splitting”, which iteratively increments the number of Gaussian mixture components. The resulting set of HMMs contains about 25000 Gaussian mixture components. Finally, the HMM set is adapted on a selection of utterances from nine speakers of the training subset of the airport information corpus by using standard acoustic model adaptation methods (combination of “maximum a posteriori” (MAP) and “maximum likelihood linear regression” (MLLR) approaches, see [YEK+02]).

The preprocessing used for the extraction of acoustic feature vectors is configured with the common setting that computes 12 MFCC coefficients, the normalized signal energy and the corresponding temporal derivations of first and second order, yielding feature vectors with 39 components.

6.3 Experimental results

The presented experimental results have the major purpose to evaluate the quality of the semantic confidence estimation. This is done by the comparison with a baseline configuration of the speech interpretation system which does not make use of the estimated confidences. Table 6.3 shows the results produced with the baseline

%	Acc_{tree}^{rel}	CER_{BL}	$CER[C]$	$CER[C_{sec}]$
CO	80.9	14.4	12.8	10.1
WC	93.4	4.8	4.2	2.2
W	87.2	9.6	8.5	5.4
TOT	85.4	11.0	9.8	7.1

Table 6.4: Tree node accuracy and confidence error rate for confidence definitions C and C_{sec} for relevant tree nodes on concept (CO), word class (WC) and word (W) hierarchy level, as well as for all relevant tree nodes (TOT).

interpretation system. As expected, the obtained recognition rates depend highly on the corresponding evaluation method (see section 6.1). The evaluations based on words (Acc_{word}) and on unfiltered parse trees (Acc_{tree}) yield a lower recognition rate, because spurious errors committed on irrelevant words or tree nodes are not suppressed. The evaluation based on parse tree nodes which are relevant for information extraction allows to consider partially correct information. Therefore, the corresponding accuracy Acc_{tree}^{rel} has a higher value than the slot accuracy Acc_{slot}. Due to the attenuation of insertion and deletion errors, the F-Measure yields the highest value.

Because of the nature of off-line evaluations, it is difficult to say which evaluation measure (Acc_{tree}^{rel}, Acc_{slot} or F-Measure) is more adequate when comparing different configurations of the interpretation system. As already mentioned, it has been decided to take the accuracy of relevant tree nodes Acc_{tree}^{rel} as optimization criterion when empirically adjusting the free system parameters on the cross-validation utterance set. This parameter setting is used during the experiments on the actual test utterance set. The comparison between the baseline configuration and the system configuration which exploits the estimated confidences is then given for all presented evaluation measures.

6.3.1 Evaluation of parse tree node confidences

At first, the results for the quality of the confidence estimation will be given on the basis of the tree-matching evaluation method. In order to show that the confidence estimation for parse tree nodes works uniformly well over all hierarchical levels, the evaluation is carried out separately for tree nodes belonging to the already mentioned hierarchy level types "word", "word class" and "concept" (see section 6.1.2). Table 6.4 shows the results for the comparison of the confidence error rate CER for the baseline system and for the two presented tree node confidence definitions C and C_{sec} (see section 5.3.2). On all hierarchy levels, one can achieve a significant reduction of the confidence error rate with a uniform setting of the confidence threshold ε which has been adjusted on the cross-validation utterance set. The consideration of sub-lattice instances with the same label and intersecting time intervals with respect to the nodes of the decoded parse tree improves the confidence estimation; the extended confidence definition C_{sec} performs significantly better than the simple definition C.

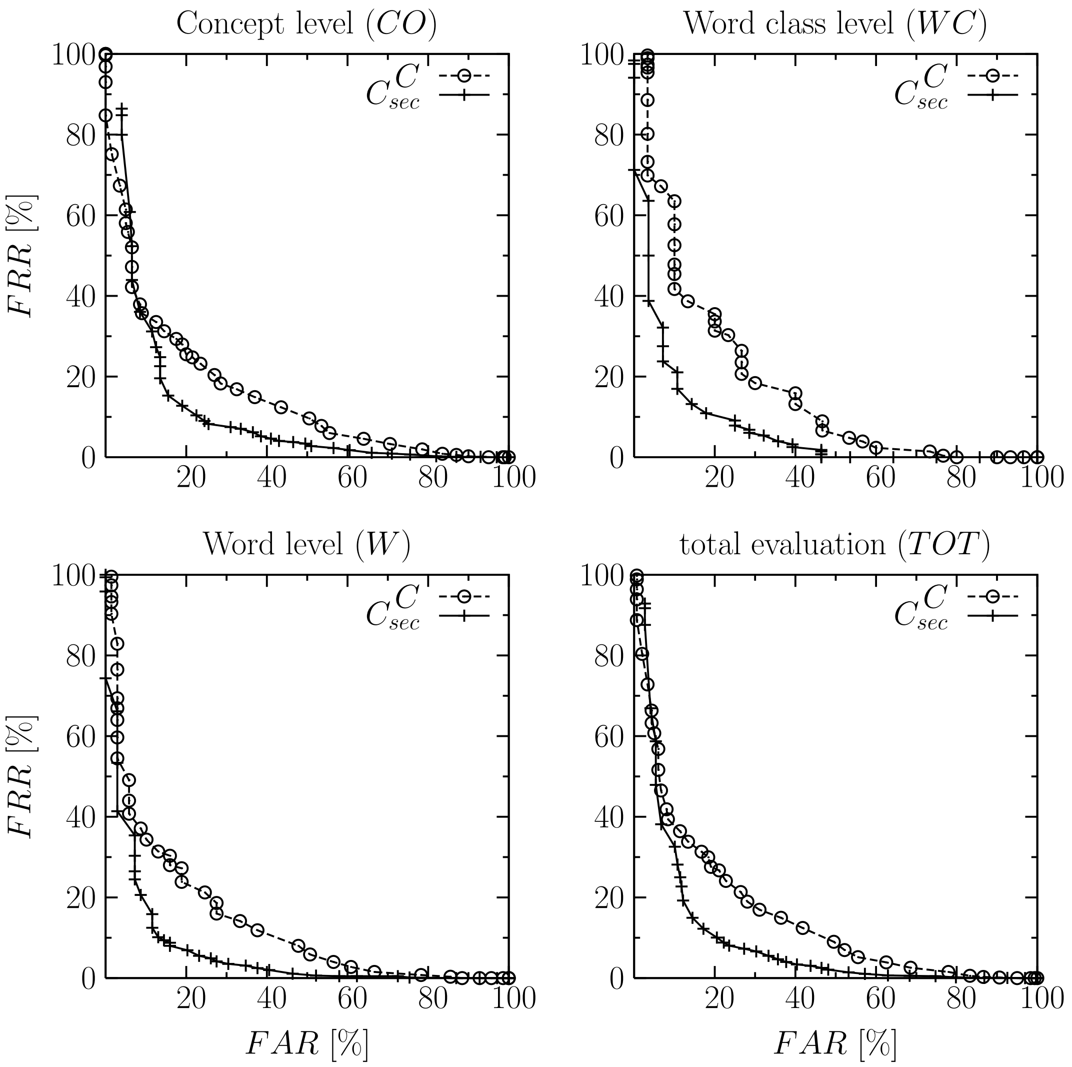

Figure 6.2: ROC-curves for confidence definitions C and C_{sec} for relevant tree nodes on concept (CO), word class (WC) and word (W) hierarchy level, as well as over all relevant tree nodes (TOT).

%	P	R	F-Measure	Acc_{slot}
Baseline	88.3	92.3	90.3	83.9
Slot-Conf.	92.1	91.2	91.6	86.3
OoV-Model	93.2	93.1	93.1	87.9

Table 6.5: Comparison of improvements in information extraction performance achieved by exploiting the estimated slot confidences and by using an explicit out-of-vocabulary model.

The obtained ROC-curves shown in figure 6.2 confirm the results of the evaluation of the confidence error rate. For all hierarchy levels, the ROC-curve of the confidence definition C_{sec} runs below the ROC-curve of definition C, at least in the area near the abscissa which is interesting for the application. When having a high recognition rate, which is the case in the experiments, the threshold ε which is adjusted to achieve a low confidence error leads to an operating point near the abscissa. For example, the setting of ε that produces the results for C_{sec} in table 6.4, leads to the operating point $\{FAR, FRR\} = \{53.2\%, 1.4\%\}$ inside the ROC-curve for all relevant tree nodes (TOT). This means, that it is possible to identify about the half of all inserted and substituted tree nodes with only rejecting a few percent of correctly decoded tree nodes.

The quality of the confidence estimation depends on the size of the generated lattices. On the one hand, the average lattice size has to be large enough to prevent a degradation of the confidence estimation. On the other hand, the computational effort during the backtracking phase of the decoding process has to be kept in reasonable limits. The size of a lattice can be expressed by the lattice density, which is the ratio of the number of lattice edges and the number of lattice nodes. By applying the lattice pruning approach discussed in section 5.3.3, the average density of the generated flat lattices has been limited to a value of about 30. This allows to effect the lattice hierarchy construction and the confidence estimation without any noticeable time delay. At the same time, there is no significant loss in the quality of the confidence estimation due to this limitation of the average lattice size.

6.3.2 Comparison with explicit out-of-vocabulary model

The evaluation of the confidence estimation shows quite encouraging results on the basis of relevant parse tree nodes. However, as explained in section 5.3.4, the tree node confidences are translated into confidences for the slot-value pairs, which are extracted from the decoded parse trees. A possible way to verify whether the improvements can also be achieved on the basis of slot-value pairs is to consider the identification of wrongly inserted slots by their bad slot confidence. If the slot confidences work well, this allows to increase the information extraction performance, because it is possible to identify more wrongly inserted slots than committing additional errors by deleting correct slots. Table 6.5 shows the comparison between the baseline information extraction performance and the performance which is obtained when deleting slots with a confidence below the confidence threshold ε. The gain of the precision rate P is greater than the loss of the recall rate R, which actually improves the obtained values for the F-Measure and for the slot accuracy Acc_{slot}.

For a better assessment of the achieved improvement, it is worthwhile to consider a system setup that includes an explicit out-of-vocabulary model which allows to identify uninterpretable parts of user utterances immediately during the one-stage decoding process. Such an experiment has been carried out using an out-of-vocabulary model which is a phoneme trigram. The trigram is estimated on the comprehensive German phonetic lexicon "Phonolex" (see [pho04]). In order to keep the computational effort during the one-stage decoding process manageable, the automaton representation of the trigram is reduced by pruning methods to a size of about 350 nodes and 800 edges. This out-of-vocabulary model has been integrated into the hierarchical language model by using the approach presented in section 3.3.3.

As one can see from table 6.5, the system with out-of-vocabulary model achieves an improvement which is about twice as high as the improvement of the system configuration which exploits the slot confidences. This is the benefit of the effort which is necessary to create and integrate the out-of-vocabulary model. In addition to its ability to identify the out-of-vocabulary words themselves, the out-of-vocabulary model also improves the recognition of the surrounding utterance parts. On the other hand, the evaluation of the estimated confidences show that it is possible to gain robustness with respect to the extracted information without the need for an additional knowledge source.

Chapter 7

Conclusion

The subject of this thesis is the task of automatic speech interpretation in the context of spoken dialog systems that provide access to information services by natural speech input. Here, automatic speech interpretation means to extract application-specific information from a user utterance that can be processed further by the spoken dialog system in order to generate the appropriate system response.

The most important resource for achieving this task provides the semantic-syntactic knowledge which describes the phrase structure of expected user utterances with respect to the application-specific information that has to be extracted. In order to represent the necessary semantic-syntactic knowledge source, this thesis adopts the commonly used approach of semantic grammars. Semantic grammars combine context-free grammar rules for the description of the phrase structure with additional rules that allow to extract application-specific information in form of slot-value pairs (semantic variables). The rules for the extraction of slot-value pairs are evaluated during the construction of the parse tree, which is done by a parser that applies the semantic grammar to the wording of a user utterance.

The major aspect of this thesis is the coupling between the task of speech recognition and the just mentioned task of semantic interpretation. A commonly used approach to solve this problem is the sequential coupling of the standard software tools speech recognizer and parser. In this setup, the speech recognizer creates an intermediate result on the basis of words, which is further processed by the parser which does the information extraction. This approach is easy to realize but has the drawback of the potential inconsistency between the knowledge sources of speech recognizer and parser, which may lead to irrecoverable interpretation errors. In order to avoid inconsistent knowledge sources, a straightforward solution is to utilize the semantic grammar as a language model which guides the speech recognition process. With the sequential coupling of speech recognizer and parser, this is usually realized by compiling the semantic grammar into a word network that can be loaded into the speech recognizer.

This thesis, however, pursues the approach to extend the word-based speech recognition algorithm in order to allow a tight coupling of speech recognition and parsing. The corresponding software tool which has been developed in the context of this thesis is called "one-stage decoder" because it is able to determine the best-matching grammatical parse tree directly from the speech signal. The one-stage decoder processes the semantic grammar in its equivalent network representation, which is a hierarchy of of transition networks that is called hierarchical language model. This model is augmented by stochastic weights which are estimated on a corpus of training utterances that has been parsed with the semantic grammar. The other knowledge sources involved in the speech recognition task, namely phonetic lexicon and acoustic models, can be represented in form of weighted transition networks, too. The resulting uniform representation of the hierarchical search space allows to determine the parse tree which best matches the recorded speech signal by applying an extended version of the Viterbi decoding algorithm which is commonly used in word-based speech recognition systems. A post-processing step applies the semantic grammar along the decoded parse tree in order to extract the contained collection of slot-value pairs that can then be passed to the dialog management module of the dialog system.

Presuming an efficient implementation, the one-stage decoder offers the advantage of a time-synchronous processing of the recorded speech signal, which may be a problem in a two-stage interpretation system, because the parser has to wait until the recognizer provides the intermediate result. Another advantage is the fast initialization of the one-stage decoder. This is the consequence of the fact that the compilation of the semantic grammar does not have to be carried out in advance, but is embedded in the underlying decoding algorithm. If requested by the dialog system, this allows a quick modification of the active semantic grammar during the time that elapses when the user is listening to the system response.

A major goal of this thesis was to provide an efficient implementation for the one-stage decoder that allows the real-time processing of user utterances on a standard desktop computer system. This goal has been achieved by employing the calculus on weighted finite-state transducers (WFSTs) in order to compile the uniform knowledge representation by the weighted transition network hierarchy into the integrated search network which is required to perform the Viterbi decoding algorithm. The great benefit of this technique, which already has been used with great success in the context of large vocabulary speech recognition systems, is the possibility to perform the search space compilation on-demand. This means that the integrated search network has to be constructed only partially for the "beam" of those hypotheses which are probable enough to be propagated during the decoding process. Furthermore, the involved automata operations optimize the topology of the integrated search network which leads to a significant speed-up of the decoding process. This speed-up has actually been observed during a comparison of the achieved runtime performance with respect to a previous version of the one-stage decoder which uses a static search space organization without applying the finite-state transducer calculus.

Apart from the realization of the speech interpretation task itself, another important subject is the "robustness" against the inevitable remainder of errors that occurs even in the case of very well-elaborated knowledge sources. During the complex task to determine an appropriate system response by processing the information which has been extracted from preceding user utterances, the dialog management benefits a great deal from the ability to detect interpretation errors. The knowledge about potential errors can be exploited to avoid misleading system responses caused by undetected interpretation errors, as well as unnecessary dialog steps which ask the user to confirm a piece of information that has been recognized correctly.

The present work investigates the approach to detect interpretation errors by the estimation of confidence measures. Without requiring an additional knowledge source, the confidence estimation can be carried out on a set of alternatives that is generated by the decoder in addition to the best-matching result. In conventional speech recognition systems this method has successfully been used to determine confidences for each recognized word via the estimation of posterior probabilities on a word lattice that represents probable alternatives to the best-matching word sequence.

During the compilation of the integrated search network, the one-stage decoder explicitly preserves the underlying structure of the semantic grammar. This has been exploited to extend the algorithm for the generation of word lattices such that it is possible to construct a set of probable alternatives to the best-matching parse tree which is compactly represented by a hierarchy of lattices. The application of the posterior probability estimation method to the representation of grammatical alternatives allows to determine confidences for every node of a decoded parse tree. The confidence of a particular parse tree node measures its reliability independently of its child nodes. By assigning the tree node confidences to the slot-value pairs which are extracted from the decoded parse tree, one can distinguish between the reliability for the type and for the value of a specific piece of extracted information, which is not possible when only considering word confidences.

The quality of the confidence estimation has been evaluated on a corpus of user utterances collected during an experiment with several test subjects that interacted with a simulated spoken dialog system offering an airport information service. This evaluation showed that the estimated confidences allow to identify about the half of all incorrectly decoded parse tree nodes with rejecting only a small percentage of the correctly decoded ones. If exploited during the information extraction by filtering out unsafely extracted slot-value pairs, the confidences yielded a significant improvement of the average information extraction performance. The magnitude of this improvement was at least half as large as the one which could be achieved when making the effort to integrate an additional knowledge source which detects uninterpretable parts of the user utterance directly during the decoding process.

The main contribution of this thesis is the development and the detailed description of the innovative one-stage decoder architecture that makes the most important features of word-based speech recognition systems, like the integrated decoding algorithm, the generation of alternatives and the estimation of confidences, available for the task of automatic speech interpretation. However, the creation of the semantic-syntactic model still requires a lot of expert knowledge in form the handcrafted semantic grammar that describes the application-specific phrase structure and the rules for extracting the corresponding information. Unfortunately, there is no consistent strategy to accomplish the task of designing a semantic grammar for a specific application domain, like the airport information service. Therefore, one has to rely on the instinct of the grammar developer, who conceives an adequate grammatical structure more or less by trail and error. Apparently, this solution has many drawbacks which make the development process of spoken dialog systems costly and very dependent on the particular application and the particular developer who designs the knowledge sources. Nevertheless, the manual design of semantic grammars is common practice, since the data-driven structural inference of semantic-syntactic models is a very demanding task that has not been solved satisfactorily, so far. Recently, interesting approaches have been published that propose to define only the set of possible semantic units and their dominance relationships. Their relation to the expected wording has to be specified only partially, particularly for the semantic units that represent information from the application data base. This knowledge and the observed set of different words defines the generic structure of an initial semantic-syntactic model. This model learns the unspecified relations with the expected wording by an iterative training of the contained stochastic weights on a corpus of domain-specific user utterances.

However, it remains to be seen how successful such approaches are in practice where the size of domain-specific corpora is often limited by the effort to conduct the necessary data collection. By all means, machine-learning approaches to semantic interpretation or even to the management of the whole dialog system have to be designed in a way that they yield controllable system configurations, which allow the systematic elimination of unacceptable errors in the resulting system behavior that occur due to the sparsity of the available data.

Bibliography

[Abn96] Steven Abney. Tagging and Partial Parsing. In K. Church, S. Young, and G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech*, Dordrecht, 1996. Kluwer Academic Publishers.

[All95] James Allen. *Natural Language Understanding.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1995.

[AM03] Cyril Allauzen and Mehryar Mohri. An Efficient Pre-Determinization Algorithm. In *Eighth International Conference on Automata (CIAA)*, Santa Barbara, July 2003.

[Baz02] I. Bazzi. *Modelling Out-of-Vocabulary Words for Robust Speech Recognition.* PhD thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Massachusetts, June 2002.

[Beu99] K. Beulen. *Phonetische Entscheidungsbäume für die Spracherkennung mit großem Vokabular.* PhD thesis, RWTH Aachen, Lehrstuhl für Informatik VI, Aachen, Germany, 1999.

[BJM83] L.R. Bahl, F. Jelinek, and R.L. Mercer. A maximum likelihood approach to continuous speech recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 5, pages 179–190, 1983.

[CG98] S.F. Chen and J. Goodman. An Empirical Study of Smooting Techniques for Language Modeling. Technical Report TR-10-98, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, August 1998.

[CRAR99] J.-C. Chappelier, M. Rajman, R. Aragües, and A. Rozenknop. Lattice Parsing for Speech Recognition. In *Proceedings of the 6th Annual Conference on Natural Language Processing (TALN 99)*, Cargèse, Corsica, July 1999.

[DG93] J. Dowding and J. Gawron. Gemini: a natural language system for spoken language understanding. In *Proceedings of the Spoken Language Systems Technology Workshop.* MIT Press, Cambridge, Massachusetts, 1993.

[DJA93] N. Dahlbäck, A. Jönsson, and L. Ahrenberg. Wizard of Oz Studies – Why and How. In *Workshop on Intelligent User Interfaces*, Orlando, Florida, 1993.

[Dup93] P. Dupont. Dynamic Use of Syntactical Knowlegde in Continuous Speech Recognition. In *Proceedings 3rd European Conference on Speech Communication and Technology*, pages 1959–1962, Berlin, Germany, September 1993.

[Ear70] J. Early. An Efficient Context-Free Parsing Algorithm. *Communications of the Association Computing Machinery*, 13(2):94–102, 1970.

[FF93] William M. Fisher and Jonathan G. Fiscus. Better Alignment Procedures for Speech Recognition Evaluation. In *Proc. ICASSP*, pages 59–62, Minneapolis, Minnesota, 1993.

[Goo53] I.J. Good. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, 40:237–264, 1953.

[GZ92] D. Goddeau and V. Zue. Integrating probabilistic LR parsing into speech understanding systems. In *Proceedings ICASSP-92*, San Francisco, California, March 1992.

[Het04] Lee Hetherington. The MIT Finite-State Transducer Toolkit for Speech and Language Processing. In *Proc. ICSLP*, Jeju Island, Korea, October 2004.

[HH00] M.E. Hennecke and G. Hanrieder. Easy Configuration of Natural Language Understanding Systems. In *Proceedings of Voice Operated Telecom Services, Do they have a bright future?*, 2000.

[HJM$^+$94] M.P. Harper, L.H. Jamieson, C.D. Mitchell, G. Ying, and et al. Integrating language models with speech recognition. In *Proceedings of the AAAI-94 Workshop on the Integration of Natural Language and Speech Processing*, Seattle, Washington, July 1994.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[HW01] K. Hacioglu and W. Ward. A Word Graph Interface for a Flexible Concept Based Speech Understanding Framework. In *Proc. Eurospeech*, Aalborg, Denmark, September 2001.

[HY05] Y. He and S. Young. Semantic Processing using the Hidden Vector State Model. *Computer Speech and Language*, 19(1):85–106, 2005.

[Jel76] F. Jelinek. Continuous Speech Recognition by Statistical Methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

[JKO01] N. Jevtic, A. Klautau, and A. Orlitsky. Estimated rank pruning and Java-based speech recognition. In *Proceedings ASRU*, Madonna di Campiglio, Italy, December 2001.

[JM99] V.M. Jiminéz and A. Marzal. Computing the K Shortest Paths: A New Algorithm and an Experimental Comparison. In J.S. Vitter and C.D. Zaroliagis, editors, *Algorithm Engineering, 3rd International Workshop, WAE '99, London, UK, July 19-21, 1999*, volume 1668 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 1999.

[JWT+95] D. Jurafsky, C. Wooters, G. Tajchman, J. Segal, A. Stolcke, E. Fosler, and N. Morgan. Using a Stochastic Context-Free Grammar as a Language Model for Speech Recognition. In *Proceedings ICASSP-95*, Detroit, Michigan, May 1995.

[Kat87] Slava M. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, March 1987.

[KN04] Stephan Kanthak and Hermann Ney. FSA: An Efficient and Flexible C++ Toolkit for Finite State Automata Using On-Demand Computation. In *ACL*, pages 510–517, 2004.

[KNRM02] Stephan Kanthak, Hermann Ney, Michael Riley, and Mehryar Mohri. A Comparison of Two LVR Search Optimization Techniques. In *Proceedings of the 7th International Conference on Spoken Language Processing*, Denver, Colorado, USA, September 2002. ISCA.

[KTK89] K. Kita and H. Saito T. Kawabata. HMM Continuous Speech Recognition Using Predictive LR Parsing. In *Proc. ICASSP*, pages 703–706, Glasgow, Scottland, 1989.

[LFRT04] Robert Lieb, Tibor Fabian, Günther Ruske, and Matthias Thomae. Estimation of Semantic Confidences on Lattice Hierarchies. In *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP 2004)*, pages 569–572, Jeju Island, Korea, October 2004.

[Low76] B.T. Lowerre. *The HARPY Speech Recognition System.* PhD thesis, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, 1976.

[LTR+05] Robert Lieb, Matthias Thomae, Günther Ruske, Daniel Bobbert, and Frank Althoff. A Flexible and Integrated Interface between Speech Recognition, Speech Interpretation and Dialog Management. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Eurospeech 2005)*, Lisbon, Portugal, September 2005.

[LY90] K. Lari and S.J. Young. The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[MB95] S. Miller and M. Bates. Recent progress in hidden understanding models. In *Proceedings of the ARPA Spoken Language Systems Technology*

Workshop, pages 22–25, Austin, Texas, 1995. Morgan Kaufmann Publishers, Palo Altos, California.

[MKSW99] J. Makhoul, F. Kubala, R. Schartz, and R. Weischedel. Performance Measures for Information Extraction. In *DARPA Broadcast News Workshop*, Herndon, Virginia, February 1999.

[Moh97] Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2), 1997.

[Moh02] Mehryar Mohri. Generic Epsilon-Removal and Input Epsilon-normalization Algorithms for Weighted Transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.

[MPM89] R. Moore, F. Pereira, and H. Murvit. Integrating Speech and Natural Language Processing. In *Proceedings Speech and Natural Language Workshop*, pages 243–247, Philadelphia, Pennsylvania, February 1989. Morgan Kaufmann Publishers, Inc., San Mateo, California.

[MPR98] M. Mohri, F. Pereira, and M.D. Riley. A Rational Design for a Weighted Finite-State Transducer Library. In Derick Wood and Sheng Yu, editors, *Automata Implementation, Second International Workshop on Implementing Automata, WIA '97, London, Ontario, Canada, September 18-20, 1997*, volume 1436 of *Lecture Notes in Computer Science*, pages 144–158, London, Ontario, Canada, 1998. Springer.

[MPR02] M. Mohri, F. Pereira, and M.D. Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[MR01] Mehryar Mohri and Michael Riley. A Weight Pushing Algorithm for Large Vocabulary Speech Recognition. In *Proc. Eurospeech*, Aalborg, Denmark, September 2001.

[NA94] H. Ney and X. Aubert. A Word Graph Algorithm for Large Vocabulary, Continuous Speech Recognition. In *Proceedings ICSLP*, Yokohama, Japan, September 1994.

[Ney91] H. Ney. Dynamic programming parsing for Context-Free grammars in Continous speech recognition. *IEEE Transactions on Signal Processing*, 39(2):336–340, 1991.

[Nua01] Nuance Communications, Menlo Park, California. *Nuance Speech Recognition System, Version 7.0, Grammar Developer's Guide*, 2001.

[ON93] M. Oerder and H. Ney. Word Graphs: An Efficient Interface Between Continuous Speech Recognition and Language Understanding. In *Proceedings ICASSP-93*, Minneapolis, Minnesota, April 1993.

[Ort98] S. Ortmanns. *Effiziente Suchverfahren zur Erkennung kontinuierlich gesprochener Sprache.* PhD thesis, RWTH Aachen, Lehrstuhl für Informatik VI, Aachen, Germany, 1998.

[pho04] Pronunciation Lexicon PHONOLEX. Bavarian Archive for Speech Signals, June 2004. Version 3.11.

[PR97] F. Pereira and M. Riley. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, Cambridge, Massachusetts, 1997. MIT Press.

[PT92] R. Pieraccini and E. Tzoukermann. Progress report on the chronus system: ATIS benchmark results. In *Proceedings of the DARPA Speech and Natural Language Workshop*. Morgan Kaufmann Publishers, Los Altos, California, 1992.

[Rab89] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77 2:257–285, 1989.

[Rin95] E.K. Ringger. A Robust Loose Coupling for Speech Recognition and Natural Language Understanding. Technical Report 592, University of Rochester, Computer Science Department, September 1995.

[RPM97] M. Riley, F. Pereira, and M. Mohri. Transducer composition for context-dependent network expansion. In *Proceedings of the 5th European Conference on Speech Communication and Technology*, pages 1427–1430, Rhodes, Greece, September 1997.

[Rus94] G. Ruske. *Automatische Spracherkennung: Methoden der Klassifikation und Merkmalsextraktion.* Oldenbourg-Verlag, München, Germany, 2nd, extended edition, 1994.

[Sen92] S. Seneff. TINA: a natural language system for spoken language applications. *Computational Linguistics*, 18(1), 1992.

[SGY05] J. Schatzmann, K. Georgila, and S. Young. Quantitative Evaluation of User Simulation Techniques for Spoken Dialogue Systems. In *6th SIGdial Workshop on Discourse and Dialogue*, Lisbon, Portugal, September 2005.

[SK83] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison Wesley, 1983.

[SO99] A. Sixtus and S. Ortmanns. High quality word graphs using forward-backward pruning. In *Proceedings ICASSP-99*, Phoenix, Arizona, March 1999.

[Spr03] R. Sproat. Lextools: a toolkit for finite-state linguistic analysis. Technical report, 2003.

[Sta97] H. Stahl. *Konsistente Integration stochastischer Wissensquellen zur semantischen Decodierung gesprochener Äußerungen.* PhD thesis, TU

München, Fakultät für Elektrotechnik und Informationstechnik, Munich, Germany, 1997.

[STN94] V. Steinbiss, B.H. Tran, and H. Ney. Improvements in Beam Search. In *Proceedings ICSLP*, pages 2143–2146, Yokohama, Japan, September 1994.

[Sto02] Andreas Stolcke. SRILM - An Extensible Language Modeling Toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing*, Denver, Colorado, USA, September 2002. ISCA.

[SZ97] D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In A. Apostolico and Z. Galil, editors, *Pattern Matching Algorithms*, chapter 14. Oxford University Press, 1997.

[TFLR03] Matthias Thomae, Tibor Fabian, Robert Lieb, and Günther Ruske. Tree Matching for Evaluation of Speech Interpretation Systems. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2003)*, pages 477–482, St. Thomas, U.S. Virgin Islands, November 2003.

[TFLR05] Matthias Thomae, Tibor Fabian, Robert Lieb, and Günther Ruske. Lexical Out-of-Vocabulary Models for One-Stage Speech Interpretation. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Eurospeech 2005)*, Lisbon, Portugal, September 2005.

[Tho06] M. Thomae. *Hierarchical Language Modeling for One-Stage Stochastic Interpretation of Natural Speech.* PhD thesis, TU München, Fakultät für Elektrotechnik und Informationstechnik, Munich, Germany, 2006.

[W3C04] W3C Voice Browser Working Group. Speech Recognition Grammar Specification. 2004.

[WAC03] Y. Wang, A. Acero, and C. Chelba. Is Word Error Rate a Good Indicator for Spoken Language Understanding Accuracy. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2003)*, St. Thomas, U.S. Virgin Islands, November 2003.

[Wah00] W. Wahlster, editor. *Verbmobil: Foundations of Speech-to-Speech Translation.* Springer, Berlin, Germany, 2000.

[WDA05] Y. Wang, L. Deng, and A. Acero. Spoken Language Understanding. *IEEE Signal Processing Magazine*, 22(5):16–31, 2005.

[WFS97] S. Wachsmuth, G.A. Fink, and G. Sagerer. Integration of parsing and incremental speech recognition. In *Proceedings of the 5th European Conference on Speech Communication and Technology*, pages 371–375, Rhodes, Greece, September 1997.

[Wil00] Daniel Willett. *Beiträge zur statistischen Modellierung und effizienten Dekodierung in der automatischen Spracherkennung.* PhD thesis, Gerhard-Mercator-Universität, Duisburg, Germany, 2000.

[Woo70] W.A. Woods. Transition Network Grammars for Natural Language Analysis. *Communications of the Association Computing Machinery*, 13(10):591–606, 1970.

[WSMN01] F. Wessel, R. Schlüter, K. Macherey, and H. Ney. Confidence Measures for Large Vocabulary Continuous Speech Recognition. *IEEE Transactions on Speech and Audio Processing*, 9(3), 2001.

[YEK+02] Steve Young, Gunnar Evermann, Dan Kershaw, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK Book (for HTK Version 3.2).* Cambridge University Engineering Department, 2002.

[YRT89] S.J. Young, N.H. Russell, and J.H.S. Thornton. Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems. Technical Report CUED/F–INFENG/TR.38, Cambridge University Engineering Department, July 1989.

[YWS+05] S.J. Young, J. Williams, J. Schatzmann, M. Stuttle, and K. Weilhammer. The Hidden Information State Approach to Dialogue Management. Technical Report CUED/F–INFENG/TR.544, Cambridge University Engineering Department, October 2005.