FeasPar - A Feature Structure Parser Learning to Parse Spontaneous Speech

Zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften der Fakultät für Informatik der Universität Karlsruhe (Technische Hochschule)

vorgelegte

Dissertation

von

Finn Dag Buø

aus Stockholm, Schweden

Tag der mündlichen Prüfung: 11. Juli 1996

Erster Gutachter: Professor Dr. Alexander Waibel
Zweiter Gutachter: Professor Dr. Walter F. Tichy

Abstract

Traditionally, automatic natural language parsing and translation have been performed with various symbolic approaches. Many of these have the advantage of a highly specific output formalism, allowing fine-grained parse analyses and, therefore, very precise translations. Within the last decade, statistical and connectionist techniques have been proposed to learn the parsing task in order to avoid the tedious manual modeling of grammar and malformation. How to learn a detailed output representation and how to learn to parse robustly even ill-formed input, has until now remained an open question.

This thesis provides an answer to this question by presenting a connectionist parser that needs a small corpus and a minimum of hand modeling, that learns, and that is robust towards spontaneous speech and speech recognizer effects. The parser delivers feature structure parses, and has a performance comparable to a good hand modeled unification based parser.

The connectionist parser FeasPar consists of several neural networks and a Consistency Checking Search. The number of, architecture of, and other parameters of the neural networks are automatically derived from the training data. The search finds the combination of the neural net outputs that produces the most probable consistent analysis.

To demonstrate learnability and robustness, FeasPar is trained with transcribed sentences from the English Spontaneous Scheduling Task and evaluated for network, overall parse, and translation performance, with transcribed and speech data. The latter contains speech recognition errors. FeasPar requires only minor human effort and performs better or comparable to a good symbolic parser developed with a 2 year, human expert effort. A key result is obtained by using speech data to evaluate the JANUS speech-to-speech translation system with different parsers. With FeasPar, acceptable translation performance is 60.5 %, versus 60.8 % with a GLR* parser. FeasPar requires two weeks of human labor to prepare the lexicon and 600 sentences of training data, whereas the GLR* parser required significant human expert grammar modeling.

Presented in this thesis are the Chunk'n'Label Principle, showing how to divide the entire parsing tasks into several small tasks performed by neural networks, as well as the FeasPar architecture, and various methods for network performance improvement. Further, a knowledge analysis and two methods for improving the overall parsing performance are presented. Several evaluations and comparisons with a GLR* parser, producing exactly the same output formalism, illustrate FeasPar's advantages.

Zusammenfassung

Traditionell sind automatisches Zerteilen und Übersetzen von natürlicher Sprache mittels verschiedener symbolischer Ansätze durchgeführt worden. Viele davon haben den Vorteil von einem hoch-spezifischen Ausgabeformalismus, der sehr genaue Zerteilanalysen ermöglicht, und deshalb sehr präzise Übersetzungen erstellt. Im Laufe des letzten Jahrzehnts, sind statistische und konnektionistische Techniken vorgeschlagen worden, um die Aufgabe des Zerteilers zu lernen, um das mühsame manuelle Modellieren von Grammatiken und deren Abweichungen zu vermeiden. Wie man detaillierte Ausgabeformalismen und das Lernen von robusten Zerteilen kombinieren soll, ist bis jetzt eine offene Frage gewesen.

Diese Arbeit liefert eine Antwort zu dieser Frage und stellt einen konnektionistischen Zerteiler vor, der ein kleines Korpus und ein Minimum von Handmodelierung braucht, der lernt, und der robust gegen spontangesprochene Sprache und Spracherkennungsfehler ist. Dieser Zerteiler liefert Merkmalsstrukturen, und hat eine Performanz ähnlich wie ein guter handmodellierter unifikationsbasierter Zerteiler.

Der konnektionistische Zerteiler FeasPar besteht aus mehreren neuronalen Netzen, deren Anzahl, Architektur und weitere Parameter automatisch aus den Trainingsdaten hergeleitet werden, und eine Konsistenzüberprüfungssuche, die unter den Ausgaben der neuronalen Netze nach der wahrscheinlichsten, konsistenten Analyse sucht.

Um Lernfähigkeit und Robustheit zu demonstrieren, wird FeasPar mit transkribierten englischen Sätzen aus der Domäne spontaner Terminabsprache trainiert. Der Zerteiler wird auf den Ebenen des Netzwerkes, Gesamtzerteilers, und Übersetzungsperformanz evaluiert. Dabei werden transkribierten und gesprochenen Daten benutzt. Die Letzteren enthalten Spracherkennungsfehler. FeasPar verlangt nur eine geringfügigen menschlischen Einsatz und hat eine Leistung, die besser oder vergleichbar mit guten symbolischen Zerteilern ist, die mit langjährigem menschlischen Expertenwissen entwickelt wurden. Ein Schlüsselergebnis ist erreicht, wenn Sprachdaten benutzt werden, um das Sprachübersetzungssystem JANUS mit verschiedenen Zerteilern zu evaluieren. Mit FeasPar war die Performanz für akzeptable Übersetzungen 60.5 % gegen 60.8 % mit einem GLR*-Zerteiler. FeasPar verlangt zwei Wochen menschlicher Arbeit, um das Lexikon und 600 Sätze als Trainingsdaten aufzubereiten, während der GLR*-Zerteiler zwei Jahre Grammatikmodellierung von menschlichen Experten verlangte.

Diese Arbeit stellt das "Chunk'n'Label"-Prinzip vor, das zeigt, wie die ganze Zerteileraufgabe in viele kleine Aufgaben, die von neuronalen Netzen ausgeführt werden, zerlegt wird, außerdem die FeasPar-Architektur und mehrere Methoden für die Leistungssteigerung der Netze. Ferner werden eine Wissensanalyse und zwei Methoden für Steigerung der Gesamtleistung des Zerteilers vorgestellt. Mehrere Vergleiche mit GLR*-Zerteilern, die genau den gleichen Ausgabeformalismus produzieren, zeigen deutlich die Vorteile von FeasPar.

Acknowledgments

I gratefully acknowledge the support from the Norwegian Research Council and Deminex grant no. ST. 153/221764, Lise og Arnfinn Hejes fond, Norway, grant no 0F0368, and finally grant 413-4001-01IV101S3 from the German Federal Ministry of Education, Science, Research and Technology (BMBF) as a part of the VERBMOBIL project. Many thanks to my advisors, Prof. Alex Waibel for setting out thesis directions, and being a source of ideas and pointing out demanding goals. and Prof. Walter Tichy for helping me to initiate my Ph.D studies and providing me with guiding thoughts about scientific work.

I would like to thank my colleagues for many fruitful discussions; Thomas Polzin, Torsten Minkwitz, Michael Finke, Andreas Eisele, Petra Geutner, Uli Bodenhausen, Lutz Prechelt and all the others. Special thanks to Torsten, Thomas and Michael and to my good North American friends Edward Pillai and Dennis King for helpful comments on contents and language of this thesis. Thanks also to Noah Coccaro and other members of the CMU/CMT project team for help concerning the ILT.

Many thanks also to my new manager and colleagues at SAP, for giving me the time and mental space I needed while finishing this work.

Last, but not least, million thanks to especially my wife Gloria, but also to friends and family in Norway and Germany, for providing mental support and understanding in moments of frustration and low progress. Without their support, this work would not have been possible.

Contents

1 Introduction **1**
- 1.1 Natural Language Translation and Parsing . . . 1
- 1.2 Spontaneous Speech . . . 2
- 1.3 The FeasPar Parser . . . 2
- 1.4 Performance Measurements . . . 5
- 1.5 Key Contributions . . . 5
- 1.6 Outline . . . 5

2 Related Work **7**
- 2.1 Symbolic and Unification Based Principles . . . 7
  - 2.1.1 Handling Extragrammatical Effects . . . 10
- 2.2 Statistical Principles . . . 11
  - 2.2.1 Grammar Inference . . . 11
  - 2.2.2 Stochastic Grammars . . . 12
  - 2.2.3 Hidden Understanding Model (HUM) . . . 12
  - 2.2.4 Alignment . . . 12
  - 2.2.5 Statistical Translation . . . 13
  - 2.2.6 Example Based Translation . . . 14
  - 2.2.7 Lexical Methods . . . 14
- 2.3 Connectionist Principles . . . 15
  - 2.3.1 Representation Issues . . . 15
  - 2.3.2 Learning Symbolic Structures . . . 16
  - 2.3.3 Direct Translation . . . 16
  - 2.3.4 Non-learning Connectionist Parsers . . . 16
  - 2.3.5 Learning Connectionist Parsers . . . 17
- 2.4 Conclusion . . . 20

3 Experimental Environment and Measures **22**
- 3.1 Experimental Environment . . . 22
- 3.2 CR Database and JANUS-I . . . 23
- 3.3 The ESST Database and JANUS-II . . . 24
- 3.4 Performance Comparison: GLR* Parser . . . 25

3.4.1 PM 1: Parse Quality . . . 26
3.4.2 PM 2: Translation Quality . . . 26

4 Baseline Principle and Architecture 28
4.1 Feature Structures . . . 28
4.2 The Chunk'n'Label Principle . . . 30
4.2.1 Theoretical Limitations . . . 32
4.3 Baseline Parser Overview . . . 34
4.3.1 Lexicon . . . 38
4.3.2 Neural Architecture and Training . . . 39
4.4 The Chunker . . . 39
4.4.1 Ordinal and Cardinal Numbers . . . 39
4.5 Linguistic Feature Labeler . . . 40
4.5.1 Syntactic Labels . . . 41
4.5.2 Semantic Labels . . . 42
4.5.3 Representation Methods . . . 43
4.5.4 Results . . . 43
4.6 The Chunk Path Task . . . 44
4.7 Neural Network Improvements . . . 51
4.7.1 Initial improvements . . . 52
4.7.2 Hybrid Encoding . . . 52
4.7.3 Second Parse . . . 54
4.7.4 Statistical Microfeatures . . . 55
4.7.5 Linear - Nonlinear Connectivity (LNC) . . . 55
4.7.6 Selected Connection Structure (SCS) . . . 57
4.7.7 Extended Context . . . 58
4.8 FeasPar Baseline Performance . . . 59
4.9 Summary . . . 60

5 Cooperative Networks 62
5.1 Knowledge Sources Analysis . . . 62
5.1.1 Identifying Knowledge Sources . . . 62
5.2 Architecture . . . 63
5.2.1 Flat Feature Structures . . . 64
5.2.2 Nested Feature Structures . . . 67
5.2.3 Multiple Feature Values . . . 69
5.2.4 Initialization Values . . . 71
5.3 Experiments . . . 71
5.4 Summary . . . 73

6 Consistency Checking Search **74**
6.1 Knowledge Sources Analysis 74
6.1.1 Global Constraints 74
6.2 Architecture 75
6.2.1 Search Task 75
6.2.2 Search Complexity Precautions 75
6.3 Search Principles 76
6.3.1 Search Implementation 77
6.4 Improvements 78
6.5 Evaluation 83
6.6 Final Evaluation 84
6.6.1 Results 84
6.6.2 Comparison Comments 85

7 Evaluation **87**
7.1 Comparison with Other Approaches 87
7.1.1 Comparison with Hand Modeled Grammars 87
7.1.2 Comparison with Connectionist Parsers 88
7.2 Suitability for Various Tasks 90
7.3 Ease of Use for Non-Experts 90
7.4 Other Advantages 91

8 Conclusion **92**
8.1 Contributions of the Thesis 92
8.2 Shortcomings 93
8.3 Future Work 94

A ESST Features **96**

Bibliography **98**

Lebenslauf (in German) **110**

Chapter 1

Introduction

1.1 Natural Language Translation and Parsing

Automatic natural language translation includes the task of parsing a source language chunk[1] and the task of generating the corresponding target language chunk. Either of two translation principles are applied: the *Transfer* Principle or the *Interlingua* Principle. The Transfer Principle states that the source language analysis A is (unfortunately) not equal to the information G that the target language generator needs. Mappings specific to language pairs must *transfer* A to G, which means that (formally) the function $A \rightarrow G$ must be defined for each pair of languages. The Interlingua Principle assumes that it is possible to define a language independent meaning representation, the so-called *interlingua*. Small domains with a limited set of concepts, often enable the definition of an interlingua, whereas the Transfer Principle is rather applied to large domains. The Interlingua Principle requires only n parsers and n generators for a domain with n languages, whereas the Transfer Principle needs $(n-1)^2$ transfer components additionally.

This work will focus on *the parsing task* needed for natural language translation applying the Interlingua Principle. Traditionally, parsing has been done with various symbolic approaches, including unification-based parsers. They have the advantage of highly specific analyses and therefore very precise translations. The output from any unification-based parser is a feature structure variation, e.g. LFG, GPSG, HPSG. A feature structure is a hierarchical set of features and values. The drawback of all symbolic approaches is the need for hand modeled grammars, which have to be adapted to specific languages and domains. Especially for spoken language (as opposed to text), the major disadvantage and difficulty for the symbolic approaches lies in building the parser:

[1] In this context, a chunk can be one of the following: text, paragraph, dialog, utterance, sentence, clause, phrase or word.

it takes as input spontaneous speech, including ungrammaticality, stops, and restarts, corrupted with speech recognition errors. The parser should output a consistent analysis in a formalism usable for processing by other components.

Approaches based on statistics or neural networks have been proposed. Their advantage is learnability and robustness. However, they all have one or more of the following disadvantages or open questions:

1. Large amounts of training data (e.g. millions of sentences) are needed.
2. The parser output formalism contains too few linguistic features to be used for further language processing, and is not guaranteed to be consistent.
3. It is an open question if the parser performs satisfactory with real world data, because it has only been evaluated with highly regular data.
4. It is not clear how well the parser as a whole performs, since no clear and quantitative statement is made about overall performance.

This thesis aims at combining the strengths of symbolic and connectionist processing, while leaving out the drawbacks by designing, implementing, and evaluating a connectionist parser using feature structures as output formalism.

1.2 Spontaneous Speech

Spontaneous speech, as opposed to read speech, contains interrupts and restarts. Furthermore, phrases with low information content occur, e.g. "let me see", "well", "i dunno", "(monday is) kind of (bad)". An example dialog from the Spontaneous Scheduling Domain is shown in Figure 1.1. The Spontaneous Scheduling Domain is a negotiation situation, in which two subjects have to decide on time and place for a meeting. The subjects' calendars have conflicts, so that a few suggestions have to go back and forth before finding a time slot suitable for both.

A speech recognizer, transforming an acoustic signal into text, inevitably produces errors. The error rate is higher when the recognizer must account for sloppyness in grammar and pronunciation of spontaneous speech. Figure 1.2 shows the same dialog processed by the state-of-the-art[2] JANUS speech recognizer. As we see, the latter dialog contains more malformed expressions.

1.3 The FeasPar Parser

In this work, the connectionist parser FeasPar[3] is presented. It parses speech, either transcribed or as delivered from a speech recognizer as shown in the previous section. The parsing architecture is also robust towards irregularities such as ungrammaticality, stops, restarts, and other spontaneous speech effects, as

[2] 78 % word accuracy

[3] **FeasPar** = **Fea**ture **s**tructure **Par**ser

Person 1:
okay
so you wanted to set up an appointment
you had mentioned the ninth
but I'll be out of town from that Tuesday the ninth until Thursday the eleventh
so I was thinking Monday we could meet after two
if you're available
and if not it'll have to be next week then

Person 2:
well
then probably looks like we'll have to meet in the next week
Monday the fifteenth I'm free in the morning
Tuesday the sixteenth I'm free in the afternoon
and Wednesday the seventeenth I'm free all day
so if you can find two hours in there we should be good

Person 1:
okay I'll tell you what
Tuesday the sixteenth looks great
any time after twelve o'clock would be fine with me
so if you have a preference let me know
probably between two and four would be great

Person 2:
alright I can do that
why don't we meet from two to four say in your office

Person 1:
okay that's fine
I'll see you then
and that's Tuesday the sixteenth from two to four
thanks

Figure 1.1: Spontaneous speech dialog in transcribed form.

well as speech recognition errors, without requiring explicit modeling of these phenomena. FeasPar provides parses with high and detailed information content, by utilizing feature structures as output formalism. FeasPar only needs a small corpus of sentences and their meanings represented as feature structures, as well as a minimum of hand modeling (lexicon modeling and aligning the training corpus) to learn the parsing task. Supported by a feature structure

Person 1:
okay
see you **want** to set up an appointment
you had **lunch on** the ninth
that i'll be out of town from that tuesday the ninth until thursday the eleventh
so i was thinking monday we **can** meet after two
if you're available
and if not **i'd** have to be next week then

Person 2:
although on
well **it** looks like we'll have to meet␣ the next week
monday the fifteenth i'm free in the morning
tuesday the sixteenth i'm free in the afternoon
and wednesday the seventeenth i'm free all day
so if you can find two hours in there we should be good

Person 1:
okay **have to be out on**
tuesday the sixteenth looks great
or any time after twelve o'clock would be fine with me
so if you**'ve** ␣ preference let me know
probably between two and four would be great

Person 2:
alright i can do **it**
when we meet from two to four **judy** in your office

Person 1:
okay that's fine
i'll see you then
the next tuesday the sixteenth from two to four
thanks

Figure 1.2: Spontaneous speech dialog in speech form, i.e. processed by the state-of-the-art JAUNS speech recognizer. Speech recognition errors are highlighted.

specification, FeasPar has a performance comparable to good hand modeled symbolic grammars.

1.4 Performance Measurements

Various methods for learning the FeasPar task are evaluated throughout this work, by using real world spontaneous speech sentences. Quantitative measurements are performed on several levels, including network level, overall parser level and translation system level. The latter is conducted by running FeasPar as a part of the JANUS speech-to-speech translation system. In order to avoid biased results, four separate data sets are used for training, testing, evaluation and final evaluation.

1.5 Key Contributions

- **Learning Complex Analyses:** As mentioned above, the existing learning parsers are incapable of learning complex analysis formalisms, which is one important strength of symbolic parsers. FeasPar closes this discrepancy by demonstrating how a complex formalism such as feature structures corresponds (Chunk'n'Label Principle) to smaller learnable tasks. FeasPar also shows how these tasks can be learned by neural networks by using various connectionist techniques.

- **Consistency Checking Search:** Running a system including several neural networks means that the total error rate multiplies up. The more fine-grained the analysis has to be, the more networks will potentially make wrong decisions. Previous connectionist parsers produce a core parse without many details, so this problem is not fatal to them. Due to FeasPar's output formalism complexity, this problem has to be taken seriously. The solution is the Consistency Check Search that not only reduces the error rate, but also assures that the parse is consistent.

- **Robustness:** FeasPar is the first learnable parser being robust towards both spontaneous speech and speech recognizer errors that performs as well as good symbolic grammars written over longer time. The robustness must neither be explicitly modeled nor learned.

1.6 Outline

In Chapter 2, other relevant work within symbolic, statistical and connectionist natural language processing is discussed. The next chapter describes the quantitative measures applied throughout this work and the environment, the JANUS translation system, in which they are performed. The notion of feature structures, the Chunk'n'Label Principle, and the FeasPar baseline architecture are explained in Chapter 4, which is concluded with overall baseline evaluation results. The following chapter discusses which further knowledge sources

can be exploited to increase overall performance, and suggests the Cooperative Network Model, which has a well explainable connectionist and statistical theoretical base. Chapter 6 presents the Consistency Checking Search, which is an efficient search returning the most probable consistent feature structure. A total evaluation of FeasPar is conducted in Chapter 7, followed by a concluding chapter.

Chapter 2

Related Work

This thesis can be seen in the context of three research areas within natural language processing (NLP): symbolic and unification based NLP, statistical NLP, and connectionist NLP. Due to the amount of research in these fields, it is only possible to mention work that is directly related to this thesis.

Unfortunately, performance evaluation tasks and measures are hardly standardized within NLP. Various principles and systems are evaluated on various tasks, and only a few results are really comparable. In order to avoid confusion, only performance results that are comparable to results from this thesis will be quoted.

Up until the last decade, NLP research dealt almost exclusively with text as input modality. As speech recognition technology became successful, the focus of interest NLP research expanded into towards speech as input modality as well.

2.1 Symbolic and Unification Based Principles

Symbolic and unification based NLP sprung out of the combination of formal linguistic theories and artificial intelligence (AI) methods for symbolic processing. The symbolic and unification based systems have the advantage of being explicit and explainable in their modeling, just like in other rule based AI systems. Linguistic phenomena, available from numerous linguistic studies and descriptions, can be viewed as expert knowledge and modeled as a grammar in order to provide an NLP system. In the following, the most influential grammar theories, parsing principles, and parsers are described.

Recursive Transition Networks (RTNs) [All88a] are equivalent to Context Free Grammars (CFGs). An RTN consists of nodes and arcs. An arc corresponds to a terminal or non-terminal in a CFG and may refer to another network, which is executed when the arc is followed. The arc 'pop' terminates

a network. Since the entire state of a parse only involves three pieces of information (current position, current node, and return points), backtracking and search is easy. The Phoenix parser, described later, uses RTNs.

A frequently applied parsing principle is *chart parsing*, which utilizes a chart, an agenda, and a CFG. The chart consists of nodes and arcs, representing word boundaries and constituents, respectively. By using the CFG rules, arcs spanning over one or few nodes are connected to span over several nodes. When an arc spans all nodes, the parse is completed. During the parsing process, various arcs joining hypotheses must be evaluated. They are stored on a stack, called an agenda.

Another popular parsing technique is *case frame instantiation* [WFT89, PBA93, BBMG+94, Min95]. The original caseframe theory [Fil68] builds on the idea of deep cases, like agent, patient, location, beneficiary or instrument. A caseframe consists of a head concept and a set of cases associated in a well-defined manner with the head concept. The main strength of caseframe parsing is the integration of syntactic parsing and semantic analysis.

The above principles were developed in the '60 and the '70. In the '80 unification based grammar theories were defined and implemented in order to enhance the means of capturing and representing linguistic information. An important consequence was that parses contained more information that could be utilized by other NLP components.

Lexical Functional Grammar (LFG) [Bre82] is a context-free phrase structure grammar annotated with features. It uses unification, and produces a constituent structure (c-structure) and a functional structure (f-structure). The c-structure is a phrase structure tree that represents the surface structure and linear order. The f-structure represents the underlying grammatical relations, and is a set of features, with feature values. The feature value is either atomic, a new f-structure, or a set of values. The set of atomic values corresponding to one feature, is always finite: e.g. the feature *Gender* takes one of the atomic values {*Feminine, Masculine, Neuter*}. The major weaknesses of LFG is the fixed constituent order, and the difficulty in implementation.

General Phrase Structure Grammar (GPSG) [GKPS85b] contains 4 parts: A theory of feature structures, Phrase-structure (PS) rules, Metarules, and grammatical principles. A feature structure has the same structure as an f-structure. The PS rules are split into ID rules and LP rules. The ID rules are normal context-free rules, but do not express the order of constituents. The LP-rules express only linear precedence that are always true for the language, e.g. in German and English: Det $<$ N, meaning that a determiner comes before its noun. This is very useful in languages without fixed word order, e.g. German [Usz86a, Usz86b, Usz87]. Metarules and grammatical principles define general language independent principles.

Head-Driven Phrase Structure Grammar (HPSG) [PS87a] is an enhancement of GPSG. The most important extensions include typed feature structures, con-

junctions and disjunctions of atomic and complex feature values,[1] implications, list and set descriptions, and finally the Head-Feature Principle (HFP). The latter expresses that when a larger constituent is built from smaller ones, a particular head feature becomes the head of the new constituent, and thereby passes head information through a constituent parse tree. HPSG grammars are continuously enhanced to encounter for various linguistic phenomena, e.g. [Par92, Net92, NNP92]. HPSG grammars are also used for spoken language processing [MST+92].

The Phoenix parser [War91, IW93, WI95, LFG+95b, BBD+95, MGWW95, MGS+95] applies RTNs. It is used for speech understanding (ATIS [2]) and translation (JANUS). Translation with Phoenix works by generating target language sentences from a parse consisting of simple slots without attributes. The advantages of the simple parse formalism, RTN, are good performance for speech understanding and translation and that the grammar hand modeling effort is smaller than for unification based grammars (9 months versus 2 years for GLR*). There are two main disadvantages: first, the translations are to a high degree pattern based standardized language and contain a low degree of variation and refinement [WMG+96], and second, the limited expressiveness in the parse formalism forces the grammar developer to write highly domain dependent rules. The parser has also been ported to other languages with mixed success, as languages with rich morpho-syntax can only be modeled with difficulty [Rec93], since the parse formalism does not offer attributes.

A good implementation of an LFG-like parser and generator is *The Generalized LR Parser and Generator* [TC87, TMML88, Tr88], in which the parsing and generation grammars are augmented context-free grammars that are compiled into an augmented LR table to be used by a run-time parser based on Tomita's generalized LR parsing algorithm [Tom85, Tom87]. The advantage is the speed in compilation of grammars and in run-time. The disadvantage is the fixed constituent order as mentioned. Sikkel and Lankhorst [SL92] further improve the LR parser with an algorithm allowing parallel processing, parsing of cyclic CFGs (not possible with Tomita's algorithm) and having a better space efficiency than Tomita's algorithm. GLR grammars have been written for various applications and domains, including technical manuals translation [MNC91], read speech translation [WNM+91, WJM+91, OAM+92, WJM+92] and (spontaneous) speech translation [WAWB+94]. Therefore, the GLR system and its successors have continuously profited from a large amount of experience and expert knowledge. Phoenix and GLR* achieve similar acceptable speech translation performance, both for transcribed input (approximately 80 %) and speech recognizer input (approximately 45 %)[3] [Lav96b].

[1] As a comparison, FeasPar only allows disjunctions of atomic feature values.

[2] ATIS (Air Traffic Information System) takes speech queries from the flight information domain and outputs SQL queries to a data base.

[3] Numbers also include translation of trivial sentences, e.g. "yes", "no", "thank you", "well", "sounds good", "terrific", "okay" etc.

All symbolic and unification based systems have the advantage of explicit models, just like other rule based AI systems. Linguistic phenomena available from numerous linguistic studies and descriptions can be viewed as expert knowledge and modeled as a grammar in order to provide a NLP system. This is very valuable when analyzing natural language that is completely well-formed with respect to its linguistic description, like text. All or almost all kinds of complex constructs in a text can be parsed, as long as it is grammatical. Hence, symbolic and unification based parsers are very successful in text parsing and understanding. The disadvantage of these approaches, however, lies in the effort spent on the development, which is very time consuming and must be performed by qualified computer linguists being experts in a particular grammar formalism.

2.1.1 Handling Extragrammatical Effects

For cognitive reasons, the underlying grammar for spoken language (speech) differs from the one for written language (text). While writing a text, the writer has more time to think, and can correct his language before issuing it, than a speaker has. A written sentence is normally grammatical, and it often has a more complex structure. A spoken sentence is not always grammatical, sometimes it contains corrections like restarts, but it often contains a simpler structure. These phenomena are gradual though, depending on how much time the writer has to plan his sentence before issuing it. He may have plenty of time (less errors) or be in a hurry (more errors). A speaker may have every word well planned in advance, giving a speech (read speech), or decide in the very moment of speaking what to say (spontaneous speech).

Symbolic and unification based systems often contain a linguistic standard grammar or subgrammar for written language, which leads to problems with extragrammatical (malformed) input. Text has a lower degree of malformation than speech. Certain errors are also caused by the machine interface: For text, keyboard typing errors occur, and for speech, recognition errors occur. The latter normally causes more errors.

There are principally two ways to compensate for natural language errors: extend or change the rules (explicit modeling), and relax rules (robustness). In order to extend the grammar, the best option would be to have a complete spoken language grammar produced by descriptive linguistics, just like written language grammars almost cover wellformed written language. A few efforts within descriptive linguistics move along these lines [LR90, Lan90]. In many real-world systems, domain specific rules catching spoken phenomena are found on an empirical, sometimes unsystematical basis. Also common is to add robustness to the system towards specific extragrammatical errors or malformation in general:

- **Specific Extragrammatical Errors**: Carbonell and Hayes [CH83] describe several extragrammatical phenomena in text and strategies for han-

dling them. The caseframe parsers CASPAR and DYPAR-II demonstrate the integrations of many of these strategies. The paper [Wen93] shows the Extended GLR parser, suggesting the handling of six phenomena in text.

- **Malformation in General**: Phoenix parses only those natural language chunks that make sense, and leaves the rest unanalyzed. TINA [Sen92, TSP+95] uses RTNs augmented with syntactic and semantic features and trainable arc probabilities. It is applied to speech understanding (ATIS). GLR* [LT93] utilizes a skip mechanism to skip words that prevent a legal parse. The parse with the lowest number of skips is considered the best. An island parser [CMGS91] examines speech by starting with the words having the highest acoustic score, and incrementally include words to both sides during parse, and thereby creating islands of interpretable natural language pieces. The DELPHI system [SB92, BIS92] applies a fallback strategy, which is enabled if its normal chart-based unification grammar parser fails, by first producing a sequence of fragmentary subparses. They are passed to a Syntactic Combiner, and if that one fails, to a Frame Combiner. The Syntactic Combiner uses extended grammar rules trying to re-construct a plausible parse. The Frame Combiner utilizes a set of frequently occurring slot-filling schemes and provides a semantic interpretation.

2.2 Statistical Principles

Statistical methods in NLP rely mainly on information that can be extracted from a corpus of collected natural language. Linguistic expert knowledge is used only to a minor degree, forming the assumptions for a statistical model.

2.2.1 Grammar Inference

The first set of methods tries to infer the grammars needed from a corpora, so that linguistic expert knowledge becomes superfluous. Naumann and Schrepp [NS92] present an incremental learning algorithm used to produce a sequence of CFGs that approximates the target grammar of the corpus. In each step, a small set of sentences is selected and analyzed by a special parser that produces partial structural descriptions for sentences not covered by the actual grammar. The sentence that minimizes the inductive leap for the learner, is selected. For this sentence, several hypotheses for completing its partial structural description are formulated and evaluated. The 'best' hypothesis is then used to infer a new grammar. This process is iterated until the corpus is entirely covered by the grammar. McCandless and Glass [MG93] start with one grammar rule per training sentence. In each step, similar[4] words are replaced by a non-terminal.

[4] measure: relative entropy

All occurrences of these words are replaced by the non-terminal. Redundant rules are removed. The process must be terminated by a stop criteria, but the authors do not suggest any. Jelinek et al. [JLM+94] reformulate the grammar learning problem to learning labeling actions (right, unary, left and up) for each node in a parse tree. A labeling task based on statistical decision trees learns the labels.

2.2.2 Stochastic Grammars

The next set of methods assumes a grammar written by linguist experts, but aims at ranking the parse tree ambiguities by probability. Jelinek et al. [JJM92] present an overview of algorithms for handling probabilistic context free grammars (PCFG). A PCFG is a CFG where probabilities are added for every rule. The Inside algorithm and the CYK algorithm calculate the total probability of a given sentence. The Viterbi search finds the most probable search tree. The Inside-Outside algorithm estimates the probabilities of the rules. Corazza et al. [CMGS91] extend these algorithms to work for an island parser, where gaps must be considered. Other work [NT87, LGQ+95, Lav96b] extend beyond only adding a probability per rule, by adding a probability per entry in the internal LR parsing table. Stolcke [Sto93] shows how to add probabilities to the CFG Earley parser.

2.2.3 Hidden Understanding Model (HUM)

The HUM model [MBSI94, MBB+95] consists of tree structured meaning expressions that are viewed as a hierarchy of Hidden Markov [Rab90] state sequences ending with the state 'exit' at every hierarchical level. The normal Hidden Markov Model (HMM) assumption of only considering a history of length one is made, so that the arc probabilities of the model can be trained as for an HMM. HUM is applied to the ATIS task. Although not mentioned by the authors, the HUM model seems to have strong similarities to RTN and Phoenix (also parsing the ATIS task), where the HUM 'exit' state corresponds to the RTN 'pop' arc.

2.2.4 Alignment

A few approaches try to learn from bilingual corpora what natural language pieces (words, phrases, sentences, paragraphs) in one language L1 correspond (align) to text pieces in another language L2. Alignment is applied in statistical translation [BCP+90] and for producing bilingual lexica. Kay and Röscheisen [KR93] present an iterative algorithm for aligning words and sentences of a bilingual corpus. The input to the algorithm is only the corpus and a trivially derived table. First, a hypothesis is assumed on which sentences in text T1 align to which sentences in text T2. Initially, only the first and last sentences in T1 and T2 are aligned with a small number of sentences from the beginning and

end of T2. Then pairs of words (W1, W2) are considered: each word W1 in a sentence S1 in T1 is compared with every word W2 in sentences S2 in T2, where S1 may align to S2. If the distributions of W1 and W2 are sufficiently similar, then W1 and W2 are assumed to align. Based on word alignment hypotheses, new sentence alignment hypotheses are calculated by considering how well the words in T1 align with words in T2. Now, a new iteration step is made. The method is tested with a scientific article available in English and German.

Gale and Church [GC93] do sentence alignment by assuming that longer sentences in T1 align to longer sentences in T2, and that shorter sentences in T1 align to shorter sentences in T2. A probabilistic score is assigned to each pair of proposed sentence pairs, based on the ratio of lengths of the two sentences (in characters) and the variance of this ratio. This probabilistic score is used in a dynamic programming framework in order to find the maximum likelihood alignment of sentences. The method is tested with the large Canadian Parlament corpus in English and French. The resulting sentence alignment is used to find word alignments.

Dagan et al. [DCG93] describe a variant of the work of Brown et al. [BCP+90] (see below), where the number of parameters is reduced in order to enable aligning corpora that are smaller and noisier than the Canadian Parlament corpus. The reduction is achieved by making various assumptions, such that every word in T2 aligns to one or zero words in T1 and that it is more likely that the first word in a sentence in T1 aligns to a word near the beginning of the corresponding sentence in T2. Furthermore, most and least frequent words, as well as function words are excepted from the analysis. The method is applied to induce bilingual terminology lexica from software manuals.

2.2.5 Statistical Translation

Alignment can also be used for statistical machine translation [BCP+90]. Given a sentence T in the target language, one seeks the sentence S from which the translator produced T. The chance of error is minimized by choosing the sentence S that is most probable, given T. Hence, S should be chosen so that P (S | T) is maximized. Bayes' theorem implies:

$$P(S \mid T) = \frac{P(S)P(T|S)}{P(T)}$$

Since P(T) does not depend on S, it suffices to find the S that maximizes the product P(S) P(T|S). The first factor is called the language model probability, and the second the translation probability. The latter can be calculated from the word alignments of the words in T and S. The language model probability expresses, how the words found by alignment should be ordered to make a sentence, and is implemented as an N-gram, as known from speech recognition language modeling [Jel90]. The method is tested with the Canadian Parlament corpus, and achieves an acceptable translation rate of 48 %.

Further work by Brown et al. aims at reducing translation time by dividing

long sentences [BPP+91], and increasing translation performance by statistical word-sense disambiguation [BLM91b]. The latter works by automatically finding the binary question related to a word's context that distinguishes it the most between two word senses.

2.2.6 Example Based Translation

Another method of direct translation is Example Based Translation [FI92, FSI92]. It contains two knowledge sources: first, a data base of translation examples manually picked from a bilingual corpus; and second, a concept hierarchy, containing semantic scores. To translate a sentence, first a lookup is performed in the bilingual data base to find similar translations. The best translation is found by exchanging the differing words in a similar translation. The translation distance is found by consulting the concept hierarchy, and calculating the semantic distance of the differing words. The advantage of this method is its robustness, because it uses semantic dependencies. Further, it is efficient, since the distance calculation is fast. The drawback is the manual time effort in building the two knowledge sources, and that efficiency decreases with large knowledge bases, since the translation runs sequentially.

2.2.7 Lexical Methods

Another method of finding similar words other than alignment is using a *word space* [Sch92]. Here, four-grams of letters are picked out of a running (newspaper) text. By filtering out infrequent, redundant, uninformative and the most frequent ones, the number of four-grams is reduced to 5000. Now, for every word w, a context window is defined around every occurrence of the word, and the four-grams in the context are viewed as a context representation for w. The context representations for all occurrences are then normalized and summed up to form the word representation. More formally, if $C(w)$ is the set of positions in the corpus at which w occurs and if $\varphi(f)$ is the vector representation for four-gram f, then the vector representation $\tau(w)$ of w is defined as: (the dot stands for normalization)

$$\tau(w) = \sum_{i \epsilon C(w)} \dot{\sum_{f \text{ close to } i}} \varphi(f) \tag{2.1}$$

The training material consists of 5 months of New York Times News Service. Listing nearest neighbor of the word representations, shows that words having similar semantic and syntactic properties are bundled.

In the paper [Sch93] a similar principle is presented, where word occurrences are applied instead of four-grams. The collocation matrix is reduced by singular value decomposition, and then a context window is examined. A sum is calculated as in the previous paper, giving a word representation φ for a word

in language L1. Another word representation φ' is defined by using the sum of all φ in sentences where the word occurs. By using results from alignments between sentences in L1 and L2, the author shows how representations for word in a language L2 can be found similarly. This gives representations for words in L1 and L2, and in addition to similarity within one language, also similarity between words in two languages can be found. The latter is then in principle a bilingual lexicon, and can be used for translation. The method is tested with the Canadian Parlament corpus, and the author provides plausible examples of similarities.

2.3 Connectionist Principles

Connectionist principles (neural networks) [HKP91a] allow other forms for representation of lexicon and structural information than those offered by symbolic principles. *Learning*, a central topic of all connectionist processing, is used in most connectionist NLP systems, and utilizes corpora of training examples. The latter suggests that connectionist principles are actually a subtopic of statistical principles. However, due to the similarities within connectionist principles, they are treated as a separate topic.

2.3.1 Representation Issues

Connectionist Principles allow both *localist* and *distributed* representations. Localist representation means that one item or item feature is represented by the activity of a single unit in a neural net [WP85]. Distributed representation means that items or item features are represented as a distributed pattern of activation across a number of units [HMR86]. When using localist representation, the smallest semantically interpretable part of the representation is called a *microfeature* [WP85]. The advantage of localist representation is its transparency, whereas the distributed representation is appreciated for its learnability and storage efficiency. The FGREP [MD89b] mechanism illustrates how to learn distributed input representations by extending the error back-propagation [RHW86] one step further than in normal back-propagation. FGREP is applied to a small and regular corpora, and proves to possess good learning capabilities.

Pollack [Pol88] implemented recursive structures, e.g. a stack, by using a three-layer network. The input consisted of two parts: stack and element. The hidden layer represented the stack, and the output consisted of a stack and an element. The stack representation was learned in the hidden layers. The elements used normal local representation. His model is called RAAM (Recursive Auto-Associative Memory). He also presents a further example, where a RAAM learns a context free grammar.

2.3.2 Learning Symbolic Structures

Certain research has focused on the ability of neural networks to compute symbolic structures, like context free grammars, in order to provide a basis for grammar processing. Pioneering work by Elman [Elm90, Elm91] presented a network architecture, later called *Elman Networks*. A sequence of input is presented in separate time steps to the network. Its architecture consists of the normal input, hidden, and output layers, and in addition a context layer. The latter is copied from the hidden layer at time t, and then fed into the hidden layer together with the input layer at time $t + 1$. The usage for learning structure in letter sequences, word boundaries and lexical structure is shown. The network learned to predict the next element in the input.

Berg [Ber91] applied a mixture of RAAM and Elman networks to implement the recursiveness of the head principle of constituents. Input consisted of context and word, and output of specifier, head and two complements. The error rate was from 1 to 4%, where most errors were of lexical nature. Giles et al. [LFG95a] investigate the ability to learn to differentiate between a grammatical and an ungrammatical sentence with respect to transitivity, given regular examples. Eight different networks are tested with the task, and Elman networks learn it best.

2.3.3 Direct Translation

Chrisman [Chr91] suggests combining two RAAMs into a dual-ported RAAM, so that they share the distributed representations being learned. During learning, a sentence in language L1 is presented at the input and output of one of the RAAMs, and the corresponding sentence in language L2 is presented at the input and output of the other RAAM. The system is evaluated with 216 possible English-Spanish sentence pairs that were generated from a vocabulary of 36 English and 36 Spanish words. All sentences contain only one clause, consisting of subject, verb and predicate/object. Translation performance is 75 %. The number includes only those sentences, where rephrasing into the same language succeeded.

2.3.4 Non-learning Connectionist Parsers

A few approaches suggest building parsers where the operations are performed by neural networks, and the network weights are not learned. Waltz and Pollack [WP85] present a hand-coded mutual connected network using spreading activation and lateral inhibition. The parser has levels for input, lexical microfeatures, syntactic structure, semantic and contextual information. Also included is a hand coded lexicon, with localist representation, using microfeatures. The purpose is semantic disambiguation. The main claim is that different kinds of information are easy to integrate. In another paper [KS93] the same idea is fol-

lowed, and uses four layers (Input, Lexical, Context, Syntax), a Semantic Space and a Memory to disambiguate sentences containing prepositional attachment ambiguities.[5] Another approach is to dynamically build small networks corresponding to CFG rules during a parse, and connect these with inhibitory and excitatory connections [KK93]. Wilkens and Schnelle [WS90a] suggest a mutual non-learning network using spreading activation and lateral inhibition to do chart parsing based on Earley's algorithm. By using localist representation, they make three spaces of units which represent parse list, computation of parse list and correct parse. In another paper [WS91, WS90b], they also introduced a mutual non-learning network applying spreading activation and lateral inhibition for representing feature co-occurrence restrictions as known from GPSG [GKPS85a]. Their representation of atomic feature values is similar to the one in this thesis, but they do not present how to represent structure. Henderson [Hen94] presents a neural network parser for syntactic parsing that performs the basic operations in a unification formalism. The work argues what is plausible from a linguistic, cognitive and biological perspective, but does not discuss issues such as performance and robustness.

2.3.5 Learning Connectionist Parsers

Miikkulainen and Dyer [MD89a, MD91] present a three-layer network, with learnable input and output. The network size and input and output lengths are fixed. The network module is called FGREP. A paraphrasing system is built, consisting of 4 FGREP modules. The module tasks are to map from sentence to case roles, from case roles to scripts, and then back to case roles and sentences.

Wermter and Weber [WW94, WW96] present SCREEN, a parser for spoken language consisting of five parts, where each part consists of several modules. Each module can have a symbolic program and a neural network. The speech interface part receives input from a speech recognizer as word hypotheses and provides an analysis of the syntactic and semantic plausibility of the recognized words. The category part receives words and provides basic syntactic, basic semantic, abstract syntactic and abstract semantic categories. Knowledge about words and phrases and their categories are received by the correction part, which provides knowledge about spontaneous effects, like repair or restart. The subclause part is responsible for the detection of subclause borders. Finally, the case frame part is responsible for the overall interpretation. This part receives knowledge about abstract and semantic categories of a phrase and provides the integrated interpretation. The architecture is interesting because it explicitly models fault tolerance. The system is trained and tested on the German Regensburg corpus (train information), and the Time Scheduling Task (see Section 3.3). Test results are only presented for five selected networks. They vary from 72 % to 89 % (Time Scheduling Task). Unfortunately, no overall results are

[5] Example: Susi sees Peter with the telescope.

given. Since the modules use output from other modules, the error rates must be assumed to multiply. Further, it is not stated how many networks there are in total. Clearly, there are at least five, since results for five networks are presented. Making a best-case calculation, using the quoted performance and assuming that all other networks have a perfect test performance[6], would yield 35 % performance. Finally, the parser output formalism is not described, so it is not clear if the parse information is sufficient for other NLP processing modules.

Since PARSEC is the most relevant work to this thesis, it is described in more detail. The task of PARSEC [Jai89, Jai90, JW89, JW90b, JW90a, Jai91, Jai92] is to incrementally parse incoming words into a three level structure. It assumes that the input is one sentence. It consists of six three-layer feed-forward networks: A *preprocessor* that filters alphanumeric input, two networks that split the input into phrases, and then phrases into clauses. The last three networks label the three levels (phrase, clause and the full sentence). The training set must be labeled consistently by the modeler. All networks are classification networks, and follow the same constructive learning paradigm. The lexicon uses a localist representation, where binary features are defined by the modeler. The binary features make up a bit vector. When a word has several meanings, the features are overloaded, i.e. the corresponding bit vectors are OR'ed together. This bit vector and a unique ID binary number together make up the input vector. The input vectors are presented one at a time to the network. The system calculates the dimensions of the networks, and sets up parameter files for learning.

The training process is performed by programmable constructive learning, PCL. Two key concepts in PCL are *hidden unit type* and *learning phases*. The PCL algorithm, as shown in Figure 2.1, consists of 3 nested loops. The innermost loop iterates over forward and backward propagations [RHW86]. The middle loop increases the number of connections by adding another unit to the hidden layer. The outermost loop increases the hidden unit type. The stop criteria for the innermost and middle loop is that test set performance does not increase any more. For each type of network to learn, 3 or 4 hidden unit types are specified. In general, type 0 has a low degree of connectivity, i.e. only local information and little or no context is taken into consideration. The modeled context increases with the types. The idea is to learn context-free relations before context-dependent ones.

PARSEC also has *3 learning phases*: Phase 1 performs standard back propagation. Phase 2 is entered when all tokens can be learned during an epoch. In Phase 2, the learning rate is adjusted while learning. Phase 3 is entered when only a very few tokens remain. In Phase 3, weight modifications are only made for tokens with error rates above a certain threshold. Phases and hidden unit types are increased independently.

In order to capture the time aspect of the input, a second weight *velocity*

[6] This is an unrealistically good assumption.

```
PROCEDURE PCL(NN: neural_network)

VAR
  T, E: integer;
  U: unit of type T;
  performance_epoch,
  performance_unit: array of float;

BEGIN
  FOR each hidden unit type T = 0 TO max_unit_type(NN)
    FOR each hidden unit U = 0 TO infinite
      BEGIN
        create(U);
        add U to NN;
        FOR each epoch E = 0 TO infinite
          BEGIN
            forward_backward_propagate(NN);
            performance_epoch(E) = measure(NN);
            IF performance_epoch(E) = 100 %
              return;
            IF performance_epoch(E) < performance_epoch(E-1)
              break;
          END;
        performance_unit(U) = performance_epoch(E-1);
        IF performance_unit(U) < performance_unit(U-1)
          break;
        END;
END.
```

Figure 2.1: Programmable constructive learning (PCL) algorithm: see text for further explanations.

is used in every node in addition to the normal weight. Velocity expresses the change in activation of a node. Learning is time-consuming. It generally converges well, but depends on lexicon feature diversity, consistency in the training set, and not too complex mapping between input and output. The sentence level label, called *mood*, is the most difficult to train.

PARSEC is extended also to consider the intonation of a spoken sentence: The pitch contour is smoothed and normalized, and presented as a 75 unit

vector, hooked on to the mood label, and then trained as the other networks. Performance rises significantly.

PARSEC is applied to the *Conference Registration Task* (see Section 3.2), consisting of 204 sentences based on a 400 word lexicon. Another application is the ATIS task [Jai91, PW92] which focuses more on semantic parsing. When PARSEC is integrated with JANUS-I, one problem is that PARSEC's output contains less information than the f-structure being used for further processing. When parsing English, this is solved by building a script-based mapper with defaults. It is tailored to the test set. When parsing German, building the mapper is only possible for the simplest sentences. The more complex morpho-syntax in German, and the fact that word disambiguation has to be done to find the correct f-structure semantic concepts, makes any principally correct effort impossible.

PARSEC is tested in JANUS with *recognition noise* input on the training data, after being trained in all phases 1-3. Generalization is tested after training phases 1-2, and looking at output without sending it through the mapper.

Finally, PARSEC is tested on the spontaneous speech effects, ungrammaticality, repairs, restarts and non-words, and is reported to be robust [Jai91]. However, neither non-trivial examples nor performance results are given. Among trivial examples that PARSEC managed, was Yes, that are right. (verb disagreement). As the labels did not reflect number, they are naturally not picky about input breaking a number rule.

2.4 Conclusion

Unification-based parsers have the advantage of highly specific analyses and therefore very precise translations. The drawback of all symbolic approaches is the need of hand modeled grammars, which have to be adapted to language and domain. Further, robustness issues cause the need for extra hand modeling efforts. Two parsers have been evaluated on the same task as FeasPar, ESST (see Section 3.3):

- **Phoenix** has a simpler parse formalism (RTNs) than FeasPar, consisting of hierarchical slots without attributes. Phoenix has a similar acceptable translation end-to-end performance, but the simpler parse formalism causes many standardized translations, without the variations and refinements that more expressive parse formalisms offer. Further, the limited formalism forces the grammar writer to introduce non-general rules, highly targeted towards the domain [WMG+96]. The lack of attributes causes problems when modeling languages rich in morpho-syntax, e.g. German [Rec93], whereas FeasPar is capable of learning morpho-syntactic labels. To model a Phoenix grammar also takes more time than modeling FeasPar training data and lexicon.

- **GLR*** has the advantage of a similar acceptable translation end-to-end performance, and the same output formalism as FeasPar. Therefore, it is compared with FeasPar throughout this thesis. The major drawback of a GLR* parser is the long grammar development time.

Approaches based on statistics or neural networks have been proposed. Their advantage is learnability and robustness. However, they all have one or more of the following disadvantages or open questions:

1. Large amounts of training data (e.g. millions of sentences) are needed.
2. The parser output formalism contains too few linguistic features to be used for further language processing, and is not guaranteed to be consistent.
3. It is an open question if the parser performs satisfactory with real world data, because it has only been evaluated with highly regular data.
4. It is not clear how well the parser as a whole performs, since no clear and quantitative statement is made about overall performance.

FeasPar combines ideas from the theory of feature structures used in the unification-based parsers in order to form a learnable problem, which is learned by a connectionist architecture. The latter contains elements from the PARSEC architecture, but provides far more information in the parser output. The output consistency and performance is enhanced considerably by various neural network techniques and a search that is not present in the PARSEC system. Finally, whereas PARSEC is evaluated with read speech, FeasPar's performance is measured on spontaneous speech, which is harder to analyze.

In summing up, FeasPar combines the advantages of unification based approaches with those of connectionist approaches, and leaves out the disadvantages. It delivers feature structure parses, and has a performance as good as a good hand modeled unification based parser. Further, it only needs a small corpus and a minimum of hand modeling to learn. FeasPar is robust towards spontaneous speech and speech recognizer effects.

Chapter 3

Experimental Environment and Measures

This work is aimed at producing not only principles and architectures that seem plausible, but also at working well with respect to performance. Hence, the descriptions throughout the next chapters are accompanied by performance measurement analysis.

This chapter explains the environment in which FeasPar is tested and evaluated, the domains and data sets being used, and finally the performance measure methods and their advantages and disadvantages.

3.1 Experimental Environment

The experimental environment for FeasPar is the JANUS[WNM+91, WJM+91, OAM+92, WJM+92, WAWB+94, GSB+95] speech-to-speech translation system, as shown in Figures 3.1 and 3.2. The JANUS-I domain is the Conference Registration Task (see Section 3.2 for details), whereas the JANUS-II domain is the Spontaneous Speech Task (see Section 3.3 for details). The core of the translation system, excluding speech recognizer and synthesizer, mainly consists of the GLR or GLR* parser and generator system, see Section 2.1.

The major architectural difference between the JANUS versions is that the JANUS-I parser analysis contains mainly syntactic information which have to be mapped into the semantic interlingua. The JANUS-II parser analysis contains all necessary semantic information, so that they can be used as interlingua directly, without the need for a mapping component. The mappers at the generation side are motivated analogously.

3.2 CR Database and JANUS-I

The conference registration (CR) task consists of imaginary telephone calls to a secretary's office of an international scientific conference. The callers ask questions about hotel rooms, how to get there and request registration formulas. All sentences are read speech, i.e. reading from a sheet of paper. This means that the sentences are well formed, and no spontaneous effects or phrases occurred. The database exists in English, German, and Japanese. This thesis uses the 12 dialogs of the German database. The GLR parsers and generators of JANUS-I

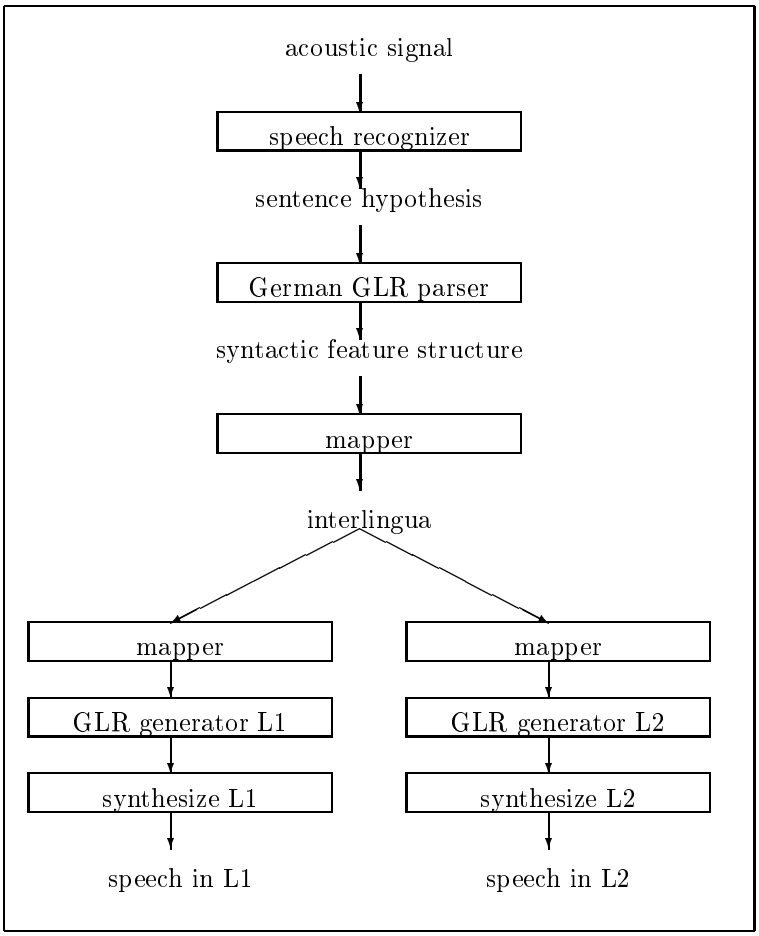

Figure 3.1: JANUS-I: speech-to-speech translation of read speech in the German Conference Registration (CR) task

are syntax-based, so that additional mappers between syntax and semantics are necessary.

3.3 The ESST Database and JANUS-II

The Spontaneous Scheduling Task (ESST) is a negotiation situation, in which two subjects have to decide on time and place for a meeting. The subjects' calendars have conflicts, so that a few suggestions have to go back and forth before finding a time slot suitable for both. The database exists in English (ESST), German (GSST), Japanese (JSST) and Spanish (SSST).

The GLR* translation system focuses on parsing ESST, and translating into the other languages. Among the generators, the English generator was the most developed. However, due to the many structurally malformed sentences in the database it is too difficult to write syntactic parsers for ESST with the

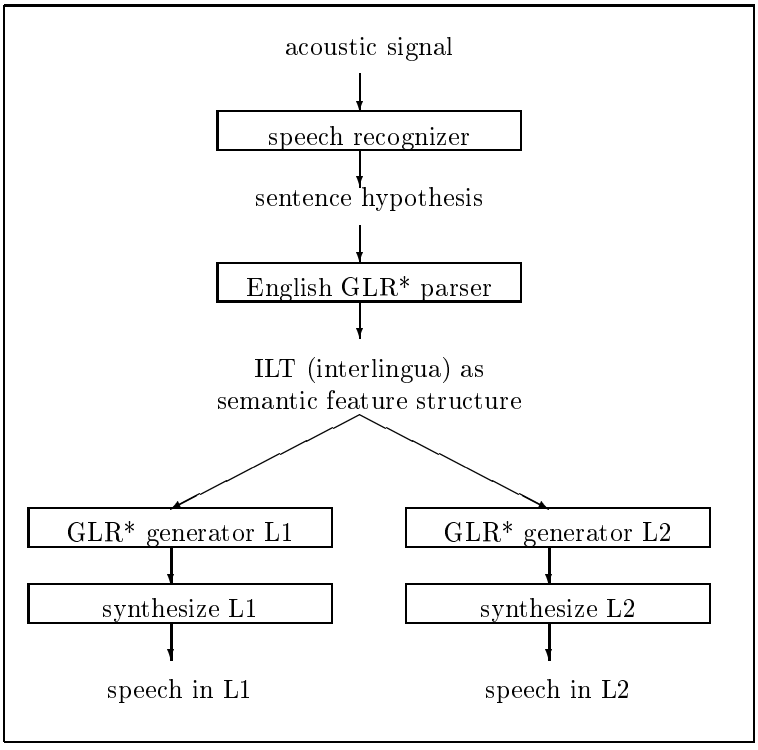

Figure 3.2: JANUS-II: speech-to-speech translation of spontaneous speech in the English Spontaneous Scheduling Task

GLR* technology. Therefore, the GLR* parsers and generators apply semantic analysis, which is used as interlingua.

The GLR* grammar and generation authors agreed on a common interlingua, ILT (InterLingua Transcription), so that the parse output could be used as generator input. During the two years of GLR* parser and generator grammar development time, the ILT was changed and extended over 50 times (!), in order to be as optimal for the GLR* parser and generators as possible. Since FeasPar is trained from data labeled with ILT segments, These labels had to be continuously manually adjusted, and the neural networks retrained when the ILT was changed. After a few times, I decided to 'freeze' the ILT for myself, and therefore conserved the ILT specification and all sentences with ILTs available by the internal JANUS-II project evaluation in March 1994. It is important to realize that FeasPar had to learn an ILT specification, which was highly tuned towards symbolic processing.

In total, this work utilizes the following ESST corpora:

Training Set (Set 1) : Approximately 560 sentences with corresponding ILTs (as of March 1994). Each sentence was manually chunked and aligned with corresponding ILT parts. The set is used for training during FeasPar development. All sentences are transcribed data (not speech data).

Test Set (Set 2) : Approximately 65 sentences with corresponding ILTs (as of March 1994). Each sentence was manually chunked and aligned with corresponding ILT parts. The set is used for testing during FeasPar development. All sentences are transcribed data (not speech data).

Evaluation Set (Set 3) : Approximately 120 sentences with corresponding ILTs (as of March 1994). are used for evaluation of the final FeasPar baseline version. The set exists in two versions: as *speech data* (with speech recognizer errors) and as *transcribed data* (without speech recognizer errors)

Final Evaluation Set (Set 4) : 99 utterances (approximately 350 sentences) without corresponding ILTs. The set is used for the final evaluation of FeasPar. This set exists as speech and transcribed data.

3.4 Performance Comparison: GLR* Parser

To show the learning ability of FeasPar, it is compared with the GLR* parser.

Since parsing spontaneous speech is more difficult and challenging than read speech, FeasPar is tested with ESST. Further, ESST offers a good chance for testing FeasPar with semantic features. Additionally, a few experiments are run with the German CR task, showing that FeasPar also can learn syntactic features (see Section 4.5.1).

On the parsing side, an ESST GLR* semantic grammar only exists for English. On the generator side, a German generator was developed only through March 94, while an English generator was continuously developed. The English grammar and generator were developed for 2 years.

3.4.1 PM 1: Parse Quality

The first performance measure, PM 1, expresses the parse quality. PM1 is also called ILT feature accuracy and is defined as:

$$\frac{C_{corr}-M}{C_{corr}}$$

where:

- M is the number of mismatches made, while checking all features in the correct ILTs *and* the suggested ILTs from the parser.
- C_{corr} is the number of considerations of existing features in the correct ILTs.

It is important to notice that this number can become negative: if a suggested ILT contains a feature A not present in the correct ILT, M is counted up, but C_{corr} is not. The latter is only incremented if feature A occurs in the correct ILT.

The advantage of PM 1 is that the measure is computed automatically and is independent of human judgement. Its disadvantage is that it is only an indirect indicator for translation quality, since not all ILT features are equally important for the generator.

3.4.2 PM 2: Translation Quality

The second performance measure, PM 2, is also called the end-to-end comparison, expresses the quality of the translated sentences. A translated sentence is graded as 'acceptable if all relevant information is conveyed and the sentence is natural (i.e. perfect), or slightly unnatural, but clear enough to understand the meaning (i.e. ok). It is graded as 'not acceptable' if incorrect or not all information is conveyed, or (with speech data only) an irrecoverable recognition error occurs. Furthermore, trivial sentences, whose translations can easily be retrieved by lookup, e.g. "yes", "no", "thank you", "well", "sounds good", "terrific", "okay" etc., are excluded, so that only truly translated sentences are counted.

Two variants exist for PM2: PM 2E is used when the parser is coupled with the English generator (developed for 2 years), and PM 2G, when the parser is combined with the German generator (development stopped March 94).

Excluded	Trivial	The sentence is trivial ("yes", "no", "okay" etc.) and excluded from evaluation
Acceptable	Perfect	Fluent translation with all information conveyed
	OK	All important information translated correctly, but translation is awkward
Not acceptable	Bad	Unacceptable translation
	Irrecoverable recognition	Translation failed due to irrecoverable speech recognition error (speech data only)

Table 3.1: Evaluation Grade Categories

The advantage of measure 2 is that translation quality is measured directly. Its disadvantage is that the grading must be done by humans, i.e. it depends on human judgement, and is therefore subjective. Also, grading becomes a fairly time consuming task.

Chapter 4

Baseline Principle and Architecture

Based on the general observations made about unification based and connectionist parsers made in chapter 2, this chapter will present a parsing method and architecture that omit the drawbacks of other parsers, as described in Section 2.4.

Section 4.1 describes the feature structure formalism. In Section 4.2 the chunk'n'label principle will be introduced and motivated. The following section describes the FeasPar baseline parser architecture, which is based on the chunk'n'label principle. A more detailed description of each of the three FeasPar baseline modules is given in Sections 4.4 - 4.6. The next section focuses on the various neural network extensions and improvements. In the last section, the final FeasPar baseline performance is presented.

4.1 Feature Structures

Many natural language processing components and applications use the feature structure formalism. It can describe all kinds of linguistic information, including discourse representation, semantics, syntax, and phonetics. When feature structures are applied in parsing, they represent the syntax or semantic analysis. Further, the they form the core of well-known unification-based formalisms e.g. LFG [KB82], GPSG [GKPS85a], and HPSG [PS87b], which all are used for a high number of parsers and natural language generators. Feature structures are used as output formalism for FeasPar. Their syntactic properties and terminology are introduced in the following:

1. A *feature structure* is a set of none, one or several *feature pairs*.

```
((speech-act *confirm)
 (sentence-type *state)
 (frame *clarify)
 (topic ((frame *simple-time)
         (day-of-week monday)))
 (adverb perhaps)
 (clarified ((frame *simple-time)
             (day-of-week monday)
             (day 27))))
```

Figure 4.1: Feature structure with the meaning "by monday i assume you mean monday the twenty seventh"

2. A *feature pair*, e.g. `(frame *clarify)` , consists of a *feature*, e.g. `frame` or `topic`, and a *feature value*.
3. A *feature value* is either:
   (a) an *atomic value*, e.g. `*clarify`[1]
   or:
   (b) a *complex value*
4. A *complex value* is a *feature structure*.

Throughout this work, examples of a certain kind of feature structures, ILTs, are presented. The following example intends to give the reader an intuitive understanding of the information contained in an ILT. The following sentence corresponds to the feature structure shown in Figure 4.1:

"by monday i assume you mean monday the twenty seventh[2]"

The semantic of the ILT in Figure 4.1 is explained briefly in the following:

- `(speech-act *confirm)`: The sentence *confirms* (and not denies or requests) a statement.
- `(sentence-type *state)`: The sentence is a *statement* (not a question or command).

[1] In the ILT specification, atomic values have an asterisk '*', and some do not. FeasPar by no means distinguishes between values with or without '*'. The examples in this work include asterisks where appropriate only in order to follow the ILT specification in every detail.

[2] All natural language examples throughout the thesis are spontaneous speech examples, and therefore presented without orthographic notation, e.g. upper case letters, commas and punctuation.

- `frame`: At each level in a feature structure the feature `frame` has one and only one value. `*clarify` means that an unclarity is clarified. `*simple-time` indicates a time expression.
- `clarified`: The information being clarified is given as the value of `clarified`.
- `topic` expresses the semantic topic or focus being emphasized in the sentence.
- `(adverb perhaps)`: An adverbial expressing uncertainty is present.
- `(day 27)`: The time expression contains the 27th day of a month.
- `(day-of-week monday)`: The weekday *monday* is part of the time expression.

ILT features and atomic values are listed in Appendix A for completeness.

4.2 The Chunk'n'Label Principle

In contrast to the standard feature structure definition of Section 4.1, an alternative view-point is to look at a feature structure as a tree of nodes and branches.[3] Each node is annotated with a set of zero, one, or several atomic feature pairs. The branches are annotated with complex features, so-called *path elements*. Each complex feature corresponds to one path element. Next, each branch is allowed to have zero, one, or several path elements. Atomic feature pairs belonging to the same branches, have the same path to all other branches.

It is assumed that there exists sentence, clause, phrase and word chunks. Then, when comparing a sentence[4] with its feature structure, it appears that there is a correspondence between parts of the feature structure, and specific chunks of the sentence. In the example feature structure of Figure 4.1, the following observations about feature pairs and paths apply:

- **feature pairs:** **corresponds to:**

  `(day 27)` "the twenty seventh"

  ```
  ((frame *simple-time)
   (day-of-week monday)
   (day 27))
  ```
  "monday the twenty seventh"

[3] This assumes that structure sharing is not possible, see Section 4.2.1.

[4] The chunk'n'label principle can easily be applied to larger parts of natural language, like e.g. an entire spoken utterance.

- **paths:** the complex value of the feature `topic` corresponds to the chunk "by monday", and the complex value of the feature `clarified` corresponds to "you mean monday the twenty seventh". Therefore, these chunks should be annotated with the paths `topic` and `clarified`, respectively.

Finally, a correspondence between chunks and nodes is defined, yielding a tree with four levels. Note that since each branch may be annotated with more than one path element, the corresponding feature structure can easily have a nesting deeper than four.

C_i	:=	'(' Q	; C_i = chunk, $i = 1..4$, Q = path
		P	; P = atomic feature pairs
		C_{i+1}^{+} ')'	; C_{i+1} = subchunk
C_5	:=	w	; single word of input sentence (terminal)
Q	:=	'[' [e('/'e)*] ']'	; e = path element (terminal)
P	:=	λ | '(' A^{+} ')'	; A = one atomic feature pair
A	:=	'(' f v ')'	; f = feature, v = atomic value (terminals)

Figure 4.2: Chunk parse: Syntax and meaning.

Manually aligning the sentence with parts of the feature structure, gives a *chunk parse*. The underlying syntax and meaning is explained in Figure 4.2. An example is shown in Figure 4.3. A few comments apply to chunk parses:

- The sentence is hierarchically split into chunks.
- Feature pairs are listed with their corresponding chunk.
- Paths are shown in square brackets, and express how a chunk relates to its parent chunk. Paths may contain more than one element. This allows several nesting levels in the corresponding feature structure.

Once having obtained the information in Figure 4.3, producing a feature structure is straight forward, using the algorithm of Figure 4.4 on Page 33.

Summing up and formalizing, the following principle, the *Chunk'n'label principle*, is introduced:

1. **Split the incoming sentence into hierarchical chunks**, see Figure 4.6.
2. **Label each chunk with feature pairs and feature paths**, see Figure 4.8.
3. **Convert this into a feature structure** (see Figure 4.9 for an example)**, using the algorithm of Figure 4.4.**

The algorithm traverses the chunk parse in a top-down manner, and builds

```
([]((speech-act *confirm)
    (sentence-type *state)
    (frame *clarify))
        ([]
                ([topic]((frame *simple-time))
                        ([]                          by)
                        ([]((day-of-week monday)) monday))
                ([]     ([]                          i))
                ([]((adverb perhaps))
                        ([]                          assume)))
        ([clarified]
                ([]     ([]                          you))
                ([]     ([]                          mean))
                ([]((frame *simple-time))
                        ([]((day-of-week monday)) monday)
                        ([]                          the)
                        ([]((day 27))                ([rego] twenty seventh)))))
```

Figure 4.3: Chunk parse: Sentence aligned with its feature structure (see text for explanation).

up a feature structure for a chunk from its path elements, atomic feature pairs, and subchunks.

4.2.1 Theoretical Limitations

The chunk'n'label principle has a few theoretical limitations compared with the feature structure formalisms commonly used in unification-based parsing, e.g. [GKPS85a]. These are discussed in the following.

Depth

With the chunk'n'label principle, the feature structure has a maximum nesting depth. One could expect the maximal nesting depth to cause limitations. However, these limitations are only theoretical, because very deep nesting is hardly needed in practice for spoken language. Due to the ability to model paths of more than length 1, no nesting depth problems occurred while modeling over 600 sentences from ESST.

```
FUNCTION start_convert(top_level_chunk: chunk): feature_structure

VAR
  S: feature_structure;
  C: chunk;

BEGIN
  S := the_empty_feature_structure;
  convert(S,top_level_chunk);
  return(S);
END;

  PROCEDURE convert(VAR S: feature_structure;
                        C: chunk);
    VAR
      E  : path_element;
      S'  : feature_structure;
      C'  : chunk;
      P,T : feature pair;

    BEGIN
      Q := chunk_path(C);
      FOR each E in Q
        BEGIN
          S' := the_empty_feature_structure;
          T  := feature_structure(E,S');
          insert T in S;
          S := S';
        END;
      FOR P in C       ; process atomic feature pairs
        insert P in S;
      FOR each C' in C ; process subchunks
        convert(S,C);
    END;
```

Figure 4.4: Top-down algorithm for converting a parse to a feature structure

Structure Sharing

Many unification formalisms allow feature values to be shared: In an example from [PS87b], p.32, subject and verb both have 3rd person-singular-feminine,

e.g. verb agreement, and they therefore have this information shared:

```
((subject   (agreement [1] ((person 3rd)
                            (number singular)
                            (gender feminine))))
 (predicate (agreement [1])))
```

The chunk'n'label principle does not incorporate any mechanism for structure sharing. The information can of course be represented in duplicated form. All work with ESST and ILT empirically showed that there is no need for structure sharing. This observation suggests that for semantic analysis, structure sharing is statistically insignificant, even if its existence is theoretically present.

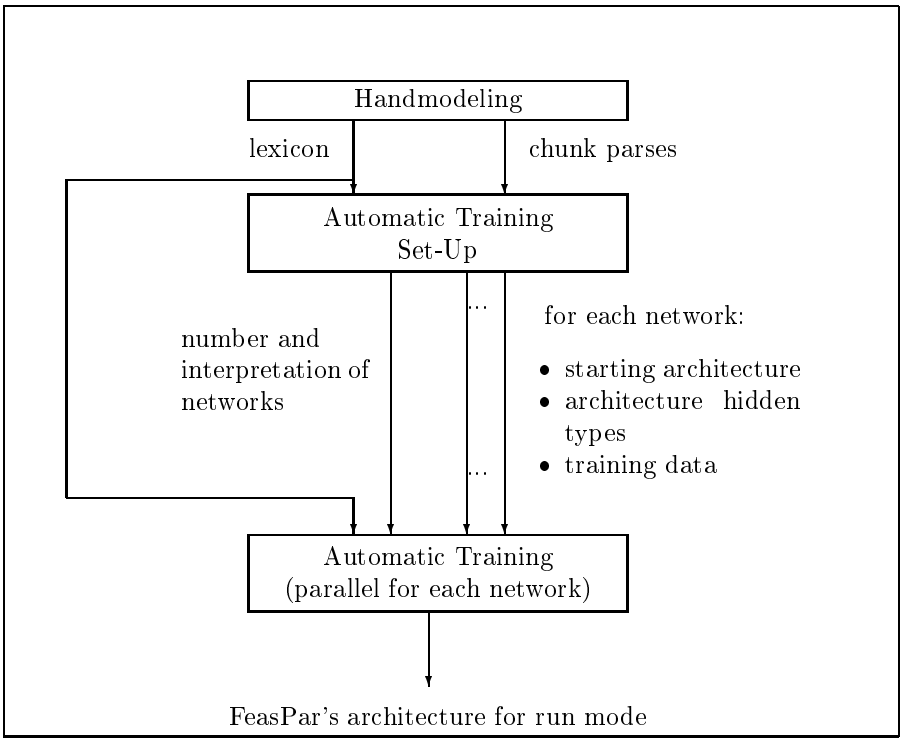

Figure 4.5: FeasPar's architecture for learn mode

4.3 Baseline Parser Overview

The chunk'n'label principle is the basis for the design and implementation of the *FeasPar* parser. This section describes the parser in overview, and the three

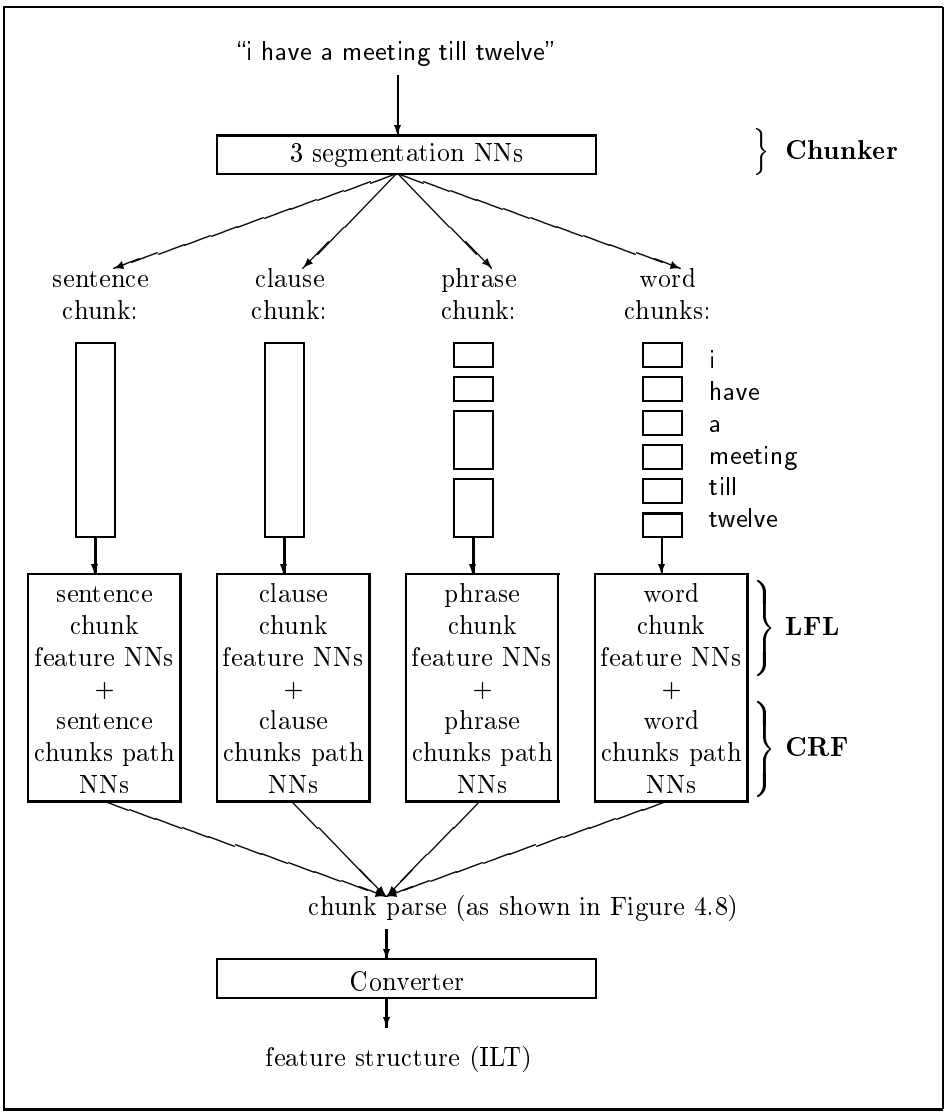

Figure 4.6: FeasPar's architecture for run mode. The Converter module varies among the different versions. In the baseline version, the algorithm in Figure 4.4 is applied, in Chapter 5, the Cooperative Networks, and in Chapter 6, the Consistency Checking Search.

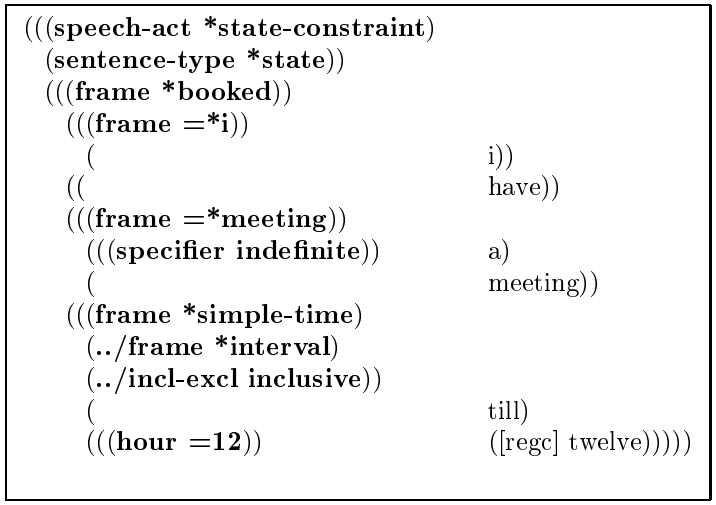

Figure 4.7: Chunked and labeled sentence (labels shown in **boldface**)

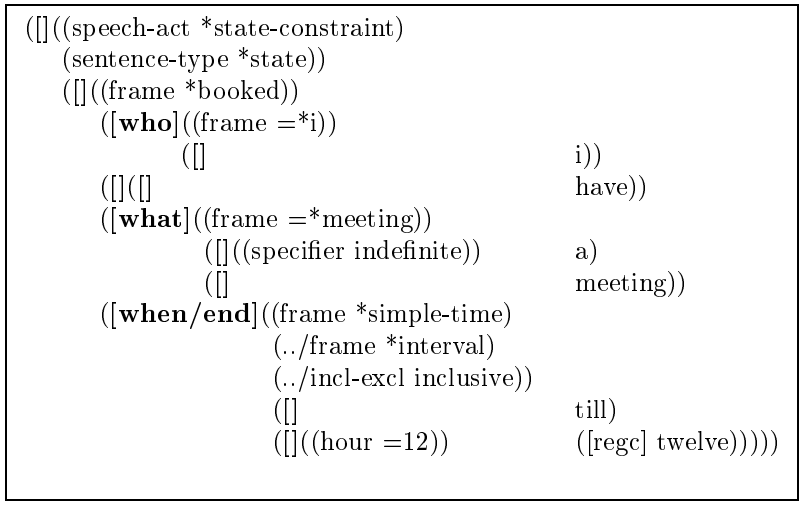

Figure 4.8: Chunk parse (chunk paths shown in **boldface**)

next sections its three main modules in more detail.

According to the chunk'n'label principle, a sentence can be chunked, and its chunks can be labeled with feature pairs and paths. This gives a chunk parse (as

```
((speech-act *state-constraint)
 (sentence-type *state)
 (frame *booked)
 (who ((frame *i)))
 (what ((frame *meeting)
        (specifier indefinite)))
 (when ((incl-excl inclusive)
        (frame *interval)
        (end ((frame *simple-time)
              (hour 12))))))
```

Figure 4.9: Feature structure parse

in Figure 4.3), which can be converted into a feature structure by the algorithm in Figure 4.4. Hence, the hard problem is to produce a chunk parse. FeasPar uses neural networks to learn to produce chunk parses. It has two modes: learn mode and run mode. In learn mode, manually modeled chunk parses are split into several separate training sets; one per neural network. Then, the networks are trained independently of each other, allowing for parallel training on several CPU's, see Figure 4.5 on Page 34. In run mode, the input sentence is processed through all networks, giving a chunk parse, which is passed on to the converting algorithm shown in Figure 4.4. The architecture and an example run is shown in Figure 4.6 on Page 35, which will both be explained in the following. FeasPar consists of several neural networks[HKP91a]. These can be grouped into three functional modules:

1. The Chunker
2. The Linguistic Feature Labeler (LFL)
3. The Chunk Path Finder (CRF)

The Chunker splits an input sentence into chunks. It consists of three neural networks. The first network finds numbers. They are classified as being ordinal or cardinal numbers, and are presented as words to the following networks. The next network groups words together to phrases. The third network groups phrases together into clauses. In total, there are four levels of chunks: word/number, phrase, clause and sentence.

The Linguistic Feature Labeler attaches features and atomic feature values (if applicable) to these chunks. A feature normally only occurs at a certain chunk level. During parsing, a neural network assigns values to features. The neural network is tailored to decide on a particular feature at a particular chunk

level. This specialization is there to prevent the learning task from becoming too complex. At each chunk level there are several features, each of them being assigned one or zero atomic feature value. Since there are many features, each chunk may get no, one or several pairs of features and atomic values. For the LFL, two different representations are suggested. These will be discussed in greater detail in Section 4.5.3. A special atomic feature value is called *lexical feature value*. It is indicated by '=' and means that the neural network only detects the *occurrence* of a value, whereas the value itself is found by a lexicon lookup. The lexical feature values are a true hybrid mechanism, where symbolic knowledge is included when the neural network signals so. Furthermore, features may be marked as *up*-features (e.g. `../incl-excl` in Figure 4.7 and 4.8 on Page 36). An up-feature is propagated up to its parent branch when building the feature structure (see Figure 4.9).

The Chunk Path Finder determines how a chunk relates to its parent chunk and consists of one network per chunk level and chunk path element.

The following example illustrates in detail how the three parts work. The parser gets the English sentence:

"i have a meeting till twelve"

The Chunker segments the sentence before passing it to the Linguistic Feature Labeler, which adds semantic labels (see Figure 4.7 on Page 36). The Chunk Path Finder then adds paths, where appropriate, and we get the complete parse as shown in Figure 4.8 on Page 36. Finally, processing it by the algorithm in Figure 4.4 on Page 33, gives the feature structure as shown in Figure 4.9.

For simplicity, this example assumes that all networks perform perfectly. The converter in Figure 4.4 only considers the output with the highest activation for every network. How to consider all outputs will the topic of chapter 5 and 6.

4.3.1 Lexicon

FeasPar uses a full word form lexicon.[5] The lexicon consists of two parts[6]: first, a syntactic and semantic microfeature [Sha91] vector per word, and second, lexical feature values.

Syntactic and semantic microfeatures are represented for each word as a vector of binary values. These vectors are used as input to the neural networks. As the neural networks learn their tasks based on the microfeatures, and not based on distinct words, adding new words using the same microfeatures is easy and does not degrade generalization performance. The number and selection of microfeatures are domain dependent and must be made manually. For CR and

[5] This means that for example the word forms "be", "is", "are", and "been" are separate lexicon entries, even if they all have the word root "be" in common.

[6] The lexicon is later extended with a third part, statistical microfeatures, see Section 4.7.4.

ESST, the lexicon contains domain independent syntactic and domain dependent semantic microfeatures. To manually model a 600 word ESST vocabulary requires 3 full days.

Lexical feature values are stored in look-up tables, which are accessed when the Linguistic Feature Labeler indicates a lexical feature value. These tables are generated automatically from the training data, and can easily be extended by hand for more generality and new words. An automatic ambiguity checker warns if similar words or phrases map to ambiguous lexical feature values.

4.3.2 Neural Architecture and Training

All neural networks have one hidden layer, and are conventional feed-forward networks. The learning is done with standard back-propagation [RHW86, HKP91b], combined with the constructive learning algorithm PCL [Jai91] (see Section 2.3.5), where learning starts using a small context, which is increased later in the learning process. This causes local dependencies to be learned first.

Generalization performance is increased by *sparse connectivity*. This connection principle is based on the microfeatures in the lexicon that are relevant to a particular network. The Chunker networks are only connected to the syntactic microfeatures, because chunking is a syntactic task. With ESST, the Linguistic Feature Labeler and Chunk Path Finder networks are connected only to the semantic microfeatures, and to relevant statistical microfeatures[7]. All connectivity setup is automatic.

4.4 The Chunker

The Chunker is almost identical to the first three PARSEC [Jai91] modules, Preprocessor, Phrase Module, and Clause Mapping Module. One extension to the Chunker is described in this section.

4.4.1 Ordinal and Cardinal Numbers

In PARSEC's preprocessor module, alphanumeric strings are detected. However, no distinction is made between cardinal and ordinal number, e.g. "three" and "third". For a task like ESST, however, this distinction is important, which should be obvious from the following example:

"I'm free at three on the third"

Here, "third" indicates the third day of the month, whereas "three" indicates three o'clock.

The Chunker uses an extended preprocessor output representation, consisting of three values: 'number', 'ordinal number', and 'cardinal number': 'Number' is used for "twenty" in "twenty nine" or "twenty ninth". 'Ordinal number' is

[7] Explained in Section 4.7.4

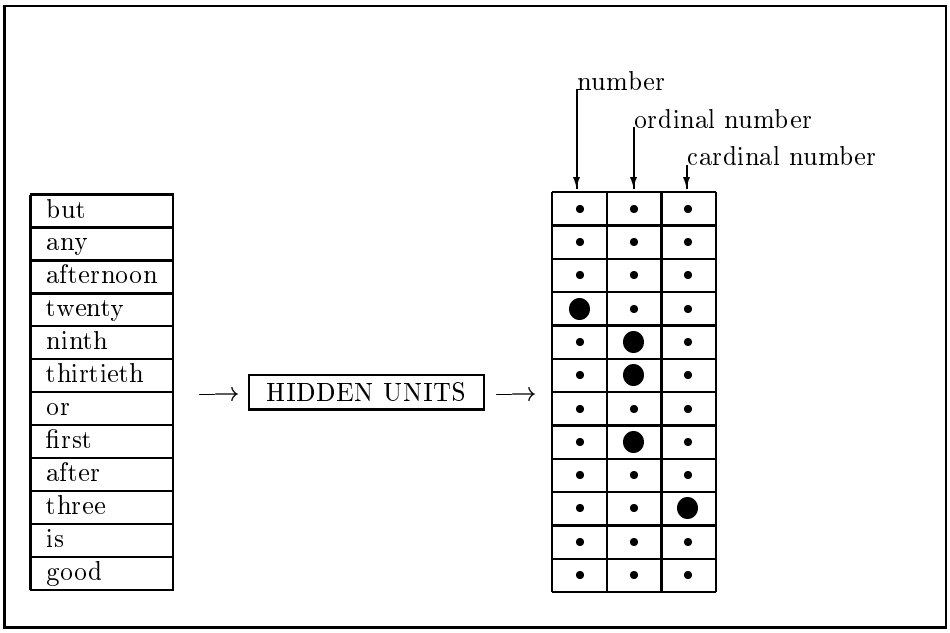

Figure 4.10: New preprocessor representation.

used for "ninth" in "twenty ninth" or "ninth". Finally, 'Cardinal number' is used for "nine" in "twenty nine" or "nine". A complete example sentence is illustrated in Figure 4.10.

4.5 Linguistic Feature Labeler

This section will discuss two aspects of the Linguistic Feature Labeler: plausibility for different kinds of parsing and representation. In order to test the plausability for FeasPar for both syntactic and semantic parsing, two different Linguistic Feature Labelers (LFL) are trained, one for German, yielding syntactic labels, and one for English, computing semantic labels. For syntactic analysis, German is chosen as a challenging example language, because of its relatively complicated morpho-syntax. This makes it a more interesting task than English syntax would have been.

```
(((form passive) (tense present)
 (mood ind) (agr sing_3))                                  ;features of verb clause 1
(                                                  falls)
 ((case nom) (agr sing_3) (gender masculine))      ihr artikel)      ;features of NP 1
                                                   akzeptiert)
(                                                  wird))
(((form active)
 (tense future) (mood ind) (agr plu_1))                    ;features of verb clause 2
(                                                  werden)
(((case nom) (agr plu_1))                          wir)              ;features of NP 2
(((case dat) (agr pol_2))                          ihnen)            ;features of NP 3
(                                                  auch)
(((case acc) (agr plu_3) (gender neuter))          spezielle formulare) ;features of NP 4
(((case acc) (agr sing_3) (gender masculine))      für ihren artikel) ;features of PP 1
(                                                  zusenden)
)
```

Figure 4.11: Syntactic labels (labels shown in **boldface**)

4.5.1 Syntactic Labels

One Chunker and an LFL are trained with the German CR task.[8] The following sentence will illustrate how parsing works (see also Figure 4.11):

> "falls ihr artikel akzeptiert wird werden wir ihnen auch spezielle formulare für ihren artikel zusenden"
>
> ("if your article accepted is will we you also special forms for your article send")
> "If your article is accepted, we will also send you special forms for your article."

First, the original sentence is chopped up into chunks. These chunks are passed to the Linguistic Feature Labeler. In this example, the Linguistic Feature Labeler faces the problem that "wird", word form of "werden", and "werden" itself have two distinct meanings (passive and future), in addition to being different word forms. For example, the `tense` feature depends both on meaning and word form. Figure 4.11 shows a parse. The feature pairs are emphasized. For completeness, Table 4.1 on Page 42 shows the features from the CR corpus, along with their meanings and value ranges.

[8]The GLR translation system for the German CR task is syntax-based. The morpho-syntactic labels used and learned by the LFL are exactly those used by the GLR translation system.

feature type	*Feature name*	*feature values*
phrase features	case	nominative, accusative, dative, genitive
	agreement	1st_singular, .., 3rd_plural, 2nd_polite
	gender	feminine, masculine, neuter
clause features	form	active, passive
	tense	present, past, perfect, pluperfect, future
	mood	indicative, subjunctive, infinitive
	agreement	1st_singular, .., 3rd_plural, 2nd_polite

Table 4.1: Syntactic labels used in German CR task

4.5.2 Semantic Labels

In order to test plausibility also for semantic labels, another Chunker and LFL is trained with ESST[9] to form a parser. The following sentence serves as an example for how the parser works:

"Can you meet in the morning"

Here the Chunker also segments the sentence before passing it to the LFL, which adds semantic labels, as shown in Figure 4.12. The complete list of semantic features is shown in Appendix A.

```
(((frame *free))
  ((                                  can))
  (((frame *you))
   (                                  you))
  ((                                  meet))
  ((frame *special-time))
   (                                  in)
   (((specifier definite))            the)
   (((time-of-day =morning))          morning)))
```

Figure 4.12: Semantic labels (labels shown in **boldface**)

[9] The GLR* translation system for ESST is semantics-based. The semantic labels used and learned by the LFL are exactly those used by the GLR* translation system.

4.5.3 Representation Methods

Two different neural network representations for the Linguistic Feature Labeler are suggested:

1. Common network representation: A network represents all features for a chunk type, so that for example all phrase chunk features are represented by one network together, and all clause chunk features in another.

2. Separate network representation: A network represents only one feature.

	syntactic features		*semantic features*	
Feature type	1. common network representation	2. separate network representation	1. common network representation	2. separate network representation
word features	n/a	n/a	97.6 %	n/a
phrase features	93.1 %	88.8 %	94.6 %	96.1 %
clause features	86.7 %	85.8 %	85.6 %	n/a
Total average	89.4 %	87.3 %	92.6 %	96.1 %

Table 4.2: Syntactic and semantic features' test set performance comparison

4.5.4 Results

This subsection discuss the test set results in the Tables 4.2 on Page 43, 4.3 on Page 44, 4.11 on Page 59, and 4.13 on Page 61 in respect to different kinds of parsing tasks and representation methods.

Syntactic and semantic parsing tasks are well mastered. The parser has learned features in German, which has with a rich and complex morpho-syntax. One word has many word forms, and each word form has many meanings. This is a very interesting problem, because the mapping from word forms to features is not trivial. Among the features, distinction can be made between simple and complex features: simple features, like `gender`, depend on one word only, and the task for them is only finding the correct feature within a word meaning. Complex features, like `tense` and `agreement` also have to combine several words that all have several meanings. For example, "werden", which has three distinct meanings, or nouns like "Artikel", which are the same in singular and plural in German. In spite of these difficulties, the LFL generalizes well. The experiments with English sentences and semantic features further confirm that the approach is suited for other kinds of features and languages. Further, all results show a tendency that features corresponding to smaller constituents are easier to

Feature type	*Feature name*	
phrase features	case	91.2 %
	agreement	94.3 %
	gender	93.8 %
	Average	93.1 %
clause features	form	91.1 %
	tense	91.4 %
	mood	83.9 %
	agreement	80.4 %
	Average	86.7 %
Total average		89.4 %

Table 4.3: Test set performance on the Linguistic Feature Labeler for syntactic features from the German CR task, broken down by features.

learn than those corresponding to larger ones. This tendency is seen for both the morpho-syntactic and the semantic features. The tendency is stronger for the semantic features, due to the lexical feature values, which are often used for semantic features, especially for those corresponding to small constituents. Lexical feature values are not used for the experiments with morpho-syntactic features described here.

The two representation methods have different advantages and disadvantages. Common network representation have the advantage of co-operation, i.e. potentially, correlations between features are exploited. Evidence for this is that total average performance for syntactic features is better with common network representation than with the separate one. Further, overall training time is shorter, and the overall network size is smaller. The separate network representation benefits from specialization. Each network is dedicated to one task only and can concentrate on that one. This is demonstrated by the results of the semantic features. The co-operation gain is hardly present for semantic features, since they do not correlate as much as syntactic features do: e.g. `exclaim` and `day` have nothing to do with each other, whereas `case` and `gender` have a strong correlation. As a consequence, the separate modules approach is included in the baseline version of the ESST parser.

4.6 The Chunk Path Task

The Chunk Path Task determines the paths of all chunks. Each chunk path element at each chunk level is represented by a neural network. By choosing the right representation of feature structures as chunk paths, the learning prob-

```
((speech-act *suggest)
 (sentence-type *directive)
 (frame *schedule)
 (what ((frame *it)))
 (when ((frame *time-list)
        (connective -)
        (items (*MULTIPLE*
                ((frame *simple-time)
                 (day 8))
                ((frame *interval)
                 (incl-excl inclusive)
                 (start ((frame *simple-time)
                         (hour 8)))
                 (end ((frame *simple-time)
                       (hour 10))))))))
 (conjunction then))
```

Figure 4.13: ILT for "then let +s plan it for then on the eighth eight to ten".

lem complexity is reduced considerably. This will be shown in this section by comparing two representation methods, *brute force* modeling and *tree* modeling.

As mentioned in Section 4.1, a chunk path may contain more than one element, in order to allow nesting depth. An example is the chunk path `when/end` in Figure 4.9 on Page 37. However, some feature structures have paths containing the path element `items`, e.g. the ILT in Figure 4.13. Since ILT models time scheduling, many time expressions appear in this style. The question is, what is the best modeling for such paths.

A *brute force* modeling, gives a training pattern as shown in Figure 4.14 on Page 46,[10] containing e.g. the chunk path `when/items/1/start`, where `items` and `1` seem necessary to determine the exact position of the atomic feature pairs of the chunk. This modeling has the problem that the chunk path of a chunk depends highly on the appearance of similar chunks in its context, and very little on the chunk content. This is apparent in Figure 4.13: only because both "on the eighth" and "eight to ten" appear in the same sentence, there is an extra step of depth containing `(frame *time-list)`.

A more consistent labeling paradigm, *the tree modeling*, models the chunks as if they appeared alone, and how sibling chunks relate to each other, i.e.

[10] The feature pair `(frame *interval)` is not modeled explicitly, because it is always triggered by the feature `incl-excl`. The same thing applies for `(frame *time-list)` and `(connective -)`, which are due to `[items]`

```
([]((speech-act *suggest)
    (sentence-type *directive))
       ([]
              ([]((conjunction =then))
                      ([]                              then))
              ([]     ([]                              let))
              ([]     ([]                              +s))
              ([]((frame =*schedule))
                      ([]                              plan))
              ([what]((frame =*it))
                      ([]                              it))
              ([when/items/1]((frame *simple-time))
                      ([]                              for)
                      ([]                              then)
                      ([]                              on)
                      ([]                              the)
                      ([]((day =8))                    ([rego] eighth)))
              ([when/items/2/start]((../incl-excl inclusive)
                                    (frame *simple-time))
                      ([]((hour =8))                   ([regc] eight)))
              ([when/items/2/end]((frame *simple-time))
                      ([]                              to)
                      ([]((hour =10))                  ([regc] ten)))))
```

Figure 4.14: Brute force modeling: training example for the sentence "then let +s plan it for then on the eighth eight to ten." Notice the usage of `when/items/1` and `when/items/2`

ignoring the `items` and `1` path elements, and model how to incrementally build a structure from separate paths. Figure 4.15 on Page 47 shows an example for how to model a training sentence. The idea behind the tree modeling is illustrated in Figures 4.16- 4.18: The chunks are viewed as loose tree branches that are incrementally added to form a tree: The first branch ("on the eighth") is created as normal, i.e. as if it were the only chunk with the path `when`. The second branch ("eight") is also labeled as if it were the only chunk with the path `when`, with the additional *control information* `new`. The latter means that this branch should be merged as a new `when` branch.

The merge result is shown in the left part of Figure 4.17. Further in the same figure, a third branch ("to ten") is also labeled as if it were the only chunk

```
([]((speech-act *suggest)
    (sentence-type *directive))
       ([]
              ([]((conjunction =then))
                     ([]                              then))
              ([]    ([]                              let))
              ([]    ([]                              +s))
              ([]((frame =*schedule))
                     ([]                              plan))
              ([what]((frame =*it))
                     ([]                              it))
              ([when]((frame *simple-time))
                     ([]                              for)
                     ([]                              then)
                     ([]                              on)
                     ([]                              the)
                     ([]((day =8))                    ([rego] eighth)))
              ([new/when/start]((../incl-excl inclusive)
                                (frame *simple-time))
                     ([]((hour =8))                   ([regc] eight)))
              ([same/when/end]((frame *simple-time))
                     ([]                              to)
                     ([]((hour =10))                  ([regc] ten)))))
```

Figure 4.15: Tree modeling: training example for the sentence "then let +s plan it for then on the eighth eight to ten". Notice the usage of `when`, `new/when/start`, and `same/when/end`, and compare with Figure 4.16 - 4.18.

with the path `[when]`, with the control information `same`. The latter means that a new branch should not be created, but that it is added to the same branch that was most recently added. The final result is shown in Figure 4.18, which is structurally equivalent to the entire `[when]` branch of the ILT in Figure 4.13 on Page 45.

In more general terms, a chunk is labeled regardless of its neighbour chunks as if they were standalones, with the addition of control information that expresses how to merge with the last chunk having the same chunk path element (e.g. `when`).

However, the control information `new` or `same` is not suffcient for an unambigious specification, as the following example will show: In Figure 4.19 on Page

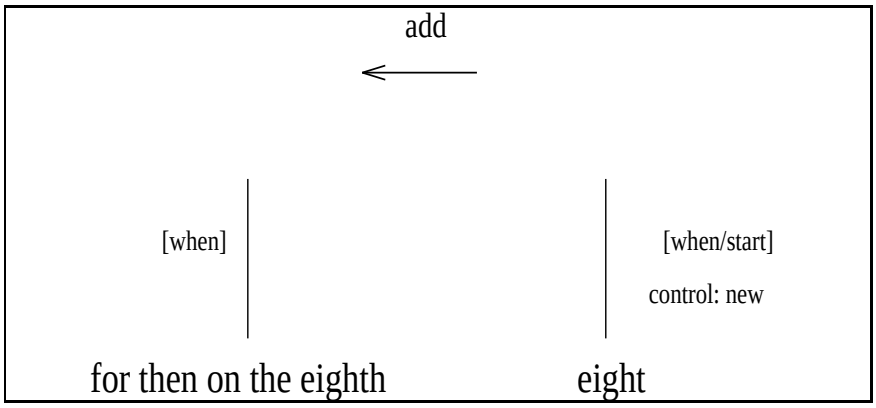

Figure 4.16: Tree modeling of chunk paths:
Left: a standalone `[when]` chunk
Right: another standalone `[when]` chunk to be merged

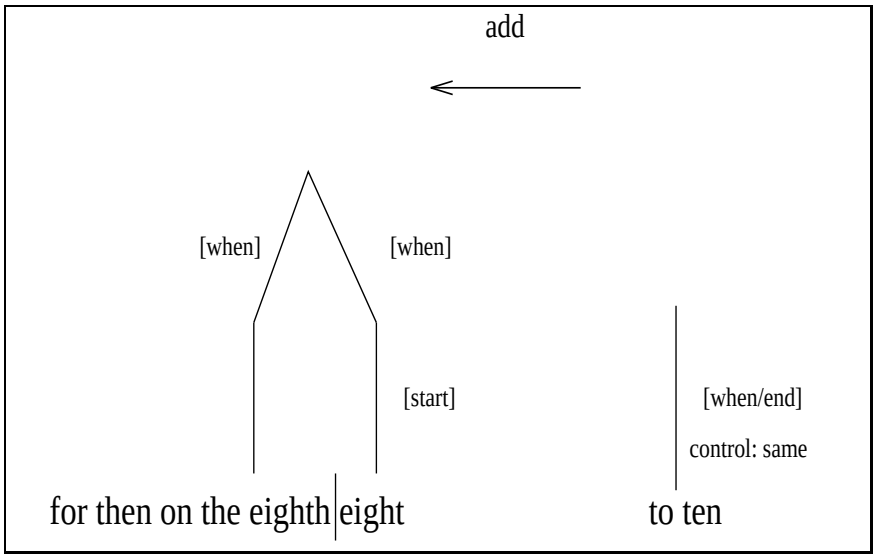

Figure 4.17: Tree modeling of chunk paths:
Left: result of merging the two first `[when]` chunks
Right: a third standalone `[when]` chunk to be merged

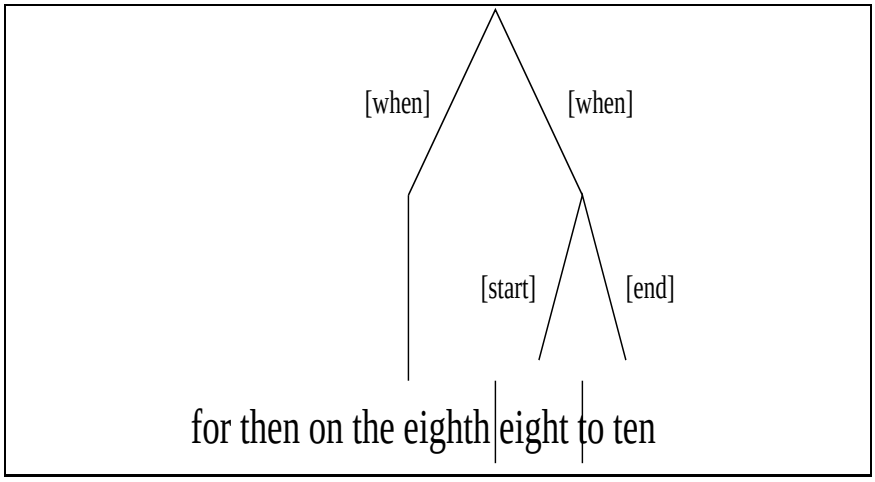

Figure 4.18: Tree modeling of chunk paths:
Result of merging all three `[when]` chunks

50 the parser must add the chunk `first` as a new branch to the parse tree. However, this is ambiguous, because it is not clear if the new branch should be a neighbor to all other branches, as in Figure 4.20 on Page 50, or if there should be a split of the previous branch, as in Figure 4.21 on Page 51. For this purpose, a control bit representing `neighbor` or `split` is necessary.

Summing up, two control bits `new` or `same` and `neighbor` or `split` for every chunk path element are in principle necessary. However, for many chunk path elements, this control information is superfluous, and receives a 'don't care' value during training. In the back-propagation algorithm, target errors calculated from 'don't care' values are not back-propagated.

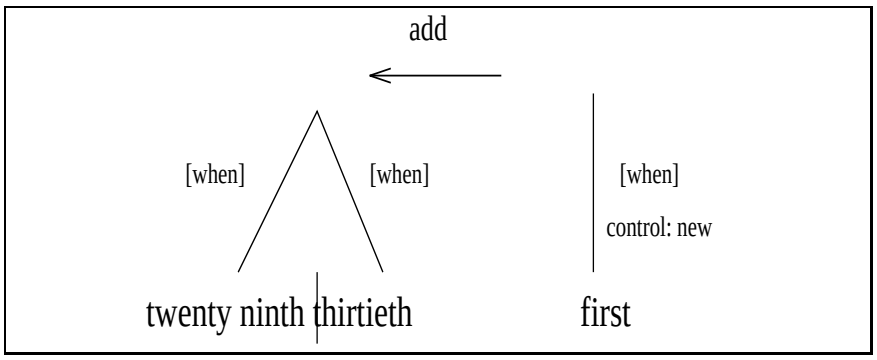

Figure 4.19: Tree modeling of chunk paths:
Ambiguous merging of the third branch.

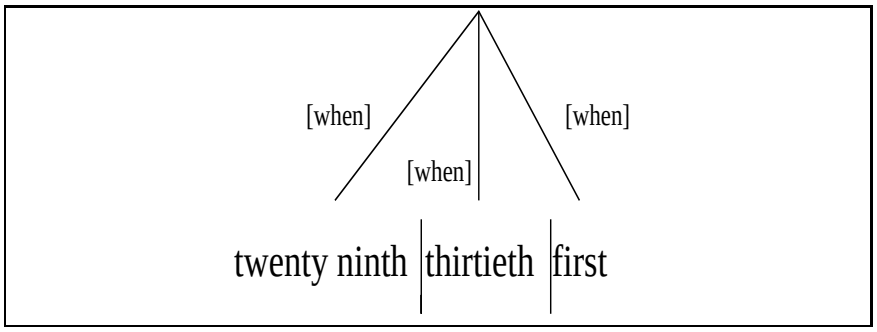

Figure 4.20: Tree modeling of chunk paths:
The third branch was included with `neighbor`.

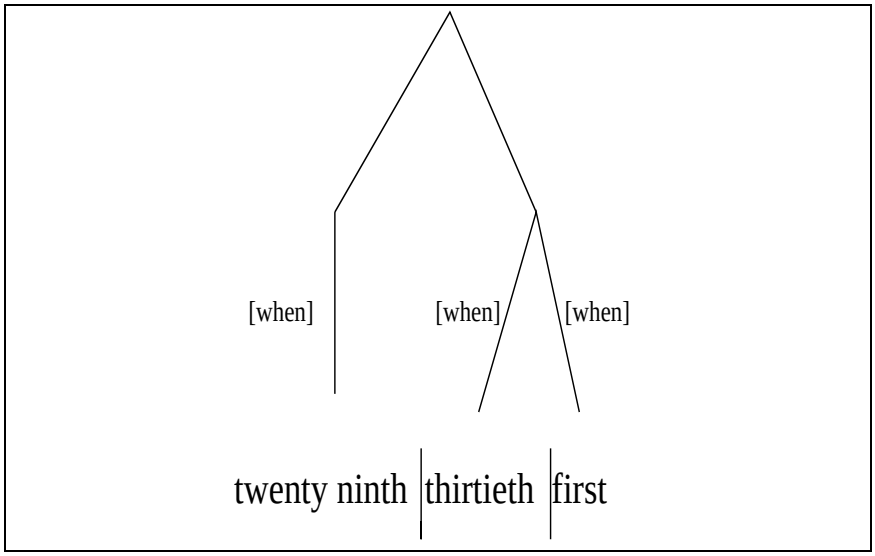

Figure 4.21: Tree modeling of chunk paths:
The third branch was included with `split`.

4.7 Neural Network Improvements

This section describes various improvements that have the potential of increasing efficiency and performance of the separate neural networks. By considering the number of networks, it is clear that the Linguistic Feature Labeler networks for word chunks and phrase chunks (LFLWP) are especially important for two reasons:

1. **Number of Networks:** The LFLWP networks make up 2/3 of all networks (29 out of 44).

2. **Usage:** For each analysis, there are more lower level chunks than upper level chunks. The LFLWP networks are applied more often than other networks. Hence, low performance in one LFLWP network would damage the final parse more than low performance in a non-LFLWP network.

Therefore, most extensions are tested on LFLWP networks. Some of the methods did not give any performance increase with ESST, but might increase performance on other tasks. One of the methods, hybrid encoding, in a reduced variant, EGREP, has been very successful (see Section 2.3.5).

4.7.1 Initial improvements

For practical reasons, neural network type of PARSEC [Jai91] is used as a starting point for the implementation. Reduction of memory usage and speed up of the training and parsing process is achieved by the following technical changes to the PARSEC software:

- **No Replication of Networks:** PARSEC replicates its networks when building a run time system after training, instead reusing networks with varying parameters. This replication has two disadvantages: Firstly, certain fixed architectural limits are introduced, e.g. constraining how many phrases could occur in a sentence. Secondly, memory is allocated excessively. FeasPar reuses networks by running them with varying parameters, thus eliminating these architectural constraints and reducing memory demand.

- **Skipping Unneccessary Unit Calculations:** For various reasons, e.g small context during early training phases, not all neural network units outputs are connected to other input units. Hence, the calculation of these outputs is not strictly necessary. Not calculating these unconnected units brings a speed up during early training of up to a factor of 5 (differs for various modules). In FeasPar, unconnected units are tracked, and excluded from calculation.

- **No Incremental Presentation of Words:** The input sentence is fed into PARSEC word by word, calculating all networks several times during this incremental feeding. The idea behind it is robustness for deletion and insertion of words. In FeasPar, all words are presented at once, so that the networks only must be run once.

- **Non-Recursive (normal) Units:** PARSEC contains a recursive unit type, where the new output calculation of a unit is a function also including the old unit output. This is motivated by the need to hold unit output values during the incremental presentation of words. In FeasPar, recursiveness is removed. Because non-recursive units have simpler derivates than recursive units, complexity is reduced and computation speed improved.

The implementation of the last two points causes LFL performance to increase slightly (see Table 4.4), and training and run times to decrease by a factor of 5 to 10, depending on sentence length.

4.7.2 Hybrid Encoding

Since neural networks have learning capabilities, one promising idea is to learn the lexicon as well, instead of having to model it by hand. This idea was first

Feature type	*Feature name*	*Incremental presentation and recursive units (PARSEC style)*	*presenting all words simultaneously and non-recursive units*
phrase features	case	91.2 %	94.3%
	agreement	94.3 %	95.8%
	gender	93.8 %	94.8%
	Average	93.1 %	95.0 %
clause features	form	91.1 %	92.9%
	tense	91.4 %	84.0%
	mood	83.9 %	91.1%
	agreement	80.4 %	78.6 %
	Average	86.7 %	86.7 %
Total average		89.4 %	90.2 %

Table 4.4: Gain of presenting input words simultaneously and using non-recursive units. Test set results on syntactic features from the German CR task.

presented by Miikkulainen [MD89b], as described in Section 2.3.5. A further advantage is that a learned lexicon can contain lexical knowledge that the human modeler does not perceive. On the other hand, one can argue that if certain lexical knowledge is obvious, it would be easy to model manually. The time effort for lexicon modeling is only a few days for a limited domain.

Based on these ideas, a *hybrid encoding* of the lexicon is suggested. The manually lexicon is expanded with one unit, which may be learned. When learning is done, and still the behavior is not perfect another one is added etc. Hence, the new units are added constructively. Since learning of the different networks occurs in parallel, the extra bits that a network trains should be visible only to that network (the first p bits remain equal for all networks, as these are the hand modeled microfeatures). If assuming n networks, the word input of network k will have the format:

$$\underbrace{\text{hand modeled microfeatures}}_{p \text{ bits}}\underbrace{\text{learned microfeatures}}_{l_k \text{ bits}}$$

where p is a constant determined before learning. l_k is 0 during the main phase of the learning, and then incremented until the network has learned perfectly.

This approach had never been tried before. Miikkulainen's FGREP [MD89b] has a fixed number of units in the lexicon that are all learnable. Hybrid encoding has two advantages over this: a) Learning goes faster when providing linguistic information. b) The number of learnable units needed are as low as possible.

For new words, the p bits are defined as described in Section 4.3.1. For the l_k bits, a simple solution is to set them to the **0**-vector. A better solution is to adopt values from similar words.

	Without hybrid encoding	With hybrid encoding
train	92.8 %	93.2 %
test	91.1 %	90.0 %

Table 4.5: Results with/without hybrid encoding

Experiments with hybrid encoding on the phrase chunking module of ESST give the results as shown in Table 4.5. They clearly indicate that hybrid encoding reduces performance. In the opinion of the author, the reason is lack of data and lack of regularity in data. Miikkulainen's sentences are highly regular, and the ESST data are very irregular. Because the approach also adds further complexity, it is not included in the final FeasPar baseline version for ESST.

4.7.3 Second Parse

The parse method described in Section 4.3 is a first parse, because each Linguistic Feature Labeler or Chunker network only has the sentence as input and decides based only on word input, without knowing the results of other classifying networks. Since the features are not independent, it would make sense to let different network decisions influence each other. The *second parse* approach enables this. First, a parse is produced as described in Section 4.3. The first parse is then presented together with the word input when the networks are run a second time. The first parse is presented as input during the second parse in the following manner:

N is a Linguistic Feature Labeler network or a Chunk Path Finder of chunk C. Then all parse results for all super- and subchunks of C are input to N.

In Table 4.6 results are shown for ESST semantic features at the phrase level. Test set performance is measured by the average over all features. Also included is the total MSE (Mean Square Error), which is a good indicator for how good the correct and incorrect learning patterns are.

test results	Without Second Parse	With Second Parse
Average phrase feature	96.1 %	95.9 %
MSE	0.210342	0.212359

Table 4.6: Results with/without Second Parse

The results clearly indicate that second parse reduces performance. In the opinion of the author, this is due to the high number of parameters to learn. Since the approach also adds further complexity (more networks), it is not included in the final FeasPar baseline version for ESST.

4.7.4 Statistical Microfeatures

Statistical microfeatures are represented for each word as a vector of continuous values v_{stat}. These microfeatures, each of them representing a feature pair (fv), are extracted automatically. For every feature value at a certain chunk level, if there exists a word such that, given this word in the training data, the feature value occurs in more than 50 % of the cases, i.e:

$$\exists \text{word } w\text{: } \frac{\#(\text{in chunk } c \text{ feature } f \text{ has value } v \wedge \text{word } w \text{ occurs in chunk } c)}{\#w} > 0.5 \wedge \#w > 1$$

One continuous microfeature value v_{stat} for a word w is set automatically to the percentage of feature value occurrence given that word w, i.e:

$$v_{stat_{w,f,v}} = \frac{\#(\text{in chunk } c \text{ feature } f \text{ has value } v \wedge \text{word } w \text{ occurs chunk } c)}{\#w}$$

In Table 4.7 results with ESST semantic features at the phrase level are shown. Test set performance is measured as average over all features. Also included is the total MSE (Mean Square Error).

test results	Without stat microfeatures	With stat microfeatures
average feature	95.3 %	95.9 %
MSE	0.231392	0.227669

Table 4.7: Results with/without statistical microfeatures

The results clearly indicate that statistical microfeatures increase performance. Therefore, it is included in the final FeasPar baseline version for ESST.

4.7.5 Linear - Nonlinear Connectivity (LNC)

The LNC idea is an extension of PARSEC's PCL idea: The principle of learning types with small context first, and extend the context in later types, is extended in the dimension 'linearity $\leftrightarrow$ nonlinearity'. Certain tasks are easy enough to be learned with linear connections (neural networks without hidden layer), and actually learn better than with nonlinear connections (neural network with hidden layer), due to the lower number of parameters. Therefore, to every nonlinear type in the learning process, a linear one with the same context is

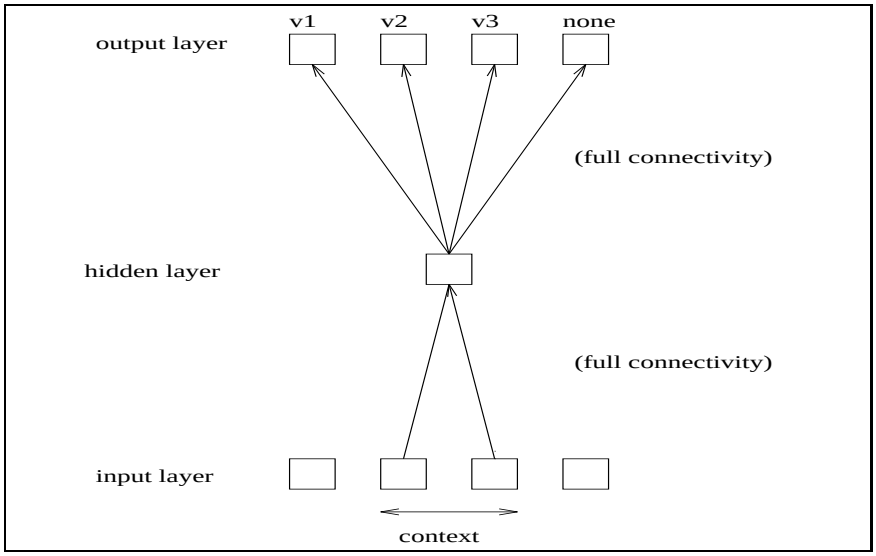

Figure 4.22: Without SCS: Output layer and hidden layer are fully connected

constructed. The learning process starts with a learning type that adds linear connections with a certain context $context_1$. In the next learning type, nonlinear connections with the same context $context_1$ are added. Then the context is increased to $context_2$, and the learning type contains linear connections with $context_2$ are added. In the next learning type, nonlinear connections with the same context $context_2$ are added, and so on.

test results	Without LNC	With LNC
frame	97.0 %	100 %
MSE	0.017471	0.003620

Table 4.8: Results with/without LNC

In Table 4.8 results are shown for ESST semantic feature `frame` for word level chunk. Test set performance is measured as average over all features. Also included is the total MSE (Mean Square Error). The results clearly indicate that LNC increases performance. Therefore, it is included in the final FeasPar baseline version for ESST.

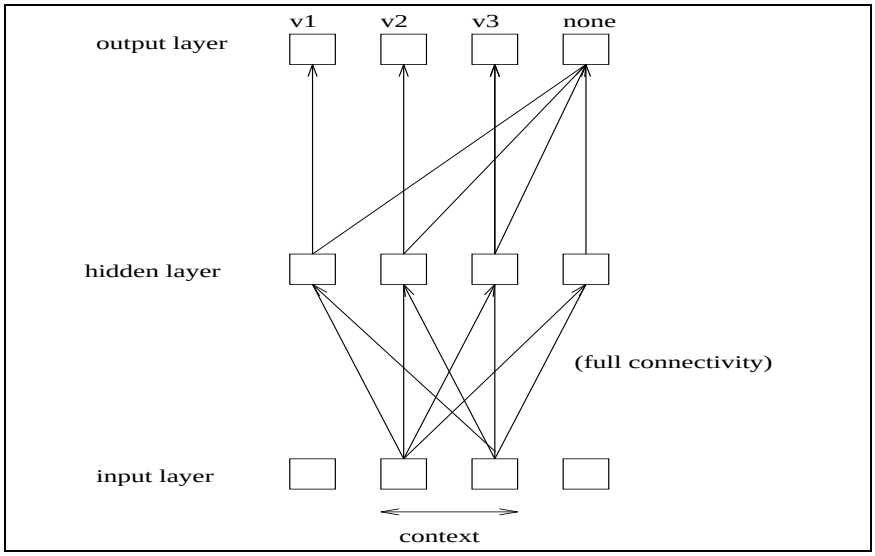

Figure 4.23: Selected connection structure (SCS): Output layer and hidden layer connected in a selective manner

test results	Without SCS		With SCS	
feature name	all	non-none-values	all	non-none-values
frame	100.0 %	100 %	100.0 %	100 %
minute	98.9 %	0 %	100.0 %	100 %
month	99.4 %	0 %	100.0 %	100 %
am-pm	100.0 %	100 %	100.0 %	100 %
hour	98.4 %	75 %	99.8 %	97 %
name	98.2 %	25 %	98.0 %	42 %
specifier	96.6 %	35 %	97.4 %	50 %

Table 4.9: Selected connection structure (SCS) performance

4.7.6 Selected Connection Structure (SCS)

Most Linguistic Feature Labeler networks for word chunks and phrase chunks (LFLWP) are uneven classification problems. This means that only 97% to 98% of the classifications should give the value `none` (i.e. no feature value for this feature), and the remaining 2 to 3 % should result in a normal feature value.

Therefore, while analyzing performance, if focus is set on these 2 to 3%

of normal feature values (non-`none` values), performance is significantly worse than for all values (see column 1 and 2 in Table 4.9). In the current network connectivity, there is full connectivity between the outputs (representing feature values) and the hidden layer (see Figure 4.22 on Page 56). This means that the hidden units are not specially dedicated to any particular part of the classification problem.

The idea of SCS is to dedicate hidden units to parts of the classification problem. The hidden unit i only has the task to decide if the ith feature value is present (output unit i high) or not (output unit low), (see Figure 4.23 on Page 57).

Table 4.9 shows results for a few example ESST semantic features for word level chunk. The results clearly indicate that SCS increases performance. Therefore, it is included in the final FeasPar baseline version for ESST.

4.7.7 Extended Context

Finally, the context growth in the learning types mentioned in Section 4.7.5 is expanded. This means that the learning process can go on beyond points where it could run out of learning types, if extended context would not be included. If performance continues to increase, it is a true gain. If not, an earlier network with better performance is chosen as the optimal architecture, and the only loss is training time. No extra performance comparison tests with extended context are presented here, since the networks in best case profit from it, and in worst case, get the same performance as without it. Extended context is included in the final FeasPar baseline version for ESST.

feature type	module name	MSE	performance
chunk paths	clauses, level 0	0.030038	97.0 %
	phrases, level 0	0.033886	78.2 %
	phrases, level 1	0.002185	97.1 %
	words, level 0	0.002902	99.8 %
chunking networks	reg	0.000307	100.0 %
	phrase	0.039171	91.2 %
	clause	0.026379	94.1 %

Table 4.10: Final baseline FeasPar for ESST: Test data set performance on chunk path and chunking networks

4.8 FeasPar Baseline Performance

A complete version of FeasPar, including the Chunker, Linguistic Feature Labeler, and the Chunk Path Finder is trained for ESST. In total, this means running training processes for 44 different networks. The test set results for all 44 networks are shown in Tables 4.13 on Page 61 and 4.10 on Page 58. Single network results are shown for completeness. This version of FeasPar is in later chapters referred to a the (final) FeasPar baseline version for ESST.

To see the cumulative performance increase of the extensions in Section 4.7,

	syntactic features		*semantic features*		
Feature type	common modules	separate modules	common modules	separate modules	separate modules **final baseline**
word features	n/a	n/a	97.6 %	n/a	**99.6 %**
phrase features	93.1 %	88.8 %	94.6 %	96.1 %	**96.9 %**
clause features	84.0 %	85.8 %	85.6 %	n/a	**92.5 %**
Total average	88.6 %	87.3 %	92.6 %	n/a	**96.3 %**

Table 4.11: Syntactic and semantic features' test set performance comparison.

PM 1:

data set	precision	normal
test	63.4%	45.2%
evaluation	58.3%	33.8%

Explanations:

- 'normal' is the measure defined in Section 3.4.1.
- 'precision' is the measure as defined in Section 3.4.1, but allows extra features and/or feature values in the output, without counting these as wrong. 'Precision' is a weaker criterion than 'normal'.

Table 4.12: ILT feature accuracy on ILT in ESST

the Linguistic Feature Labeler results are contrasted with the initial results in Table 4.11 on Page 59. It is apparent that performance increases significantly.

Finally, the parser is run as a whole to produce feature structures (ILT). FeasPar parses two different data sets. First, the test set used throughout this chapter. Second, an unseen evaluation set. Both data sets include correct ILTs. The results on the evaluation set are significantly worse than on the test set. This is not very surprising. It shows that development has been targeted towards the test set. Also, we see that the ILT performance is considerably lower than the individual networks' performance. This is due to the multiplication of individual network errors.

4.9 Summary

This chapter presented the chunk'n'label principle for mapping natural language into feature structures. The FeasPar parser baseline version was introduced. It consists of several back-propagation neural networks. New methods for improving the performance of these networks were presented and tested. The final baseline evaluation shows that each neural network performs well or very well, but that the total parse result (ILT feature accuracy) is not satisfactory.

feature type	feature name	MSE	performance
word features	am-pm	0.004702	100.0 %
	day	0.005742	100.0 %
	day-of-week	0.006374	99.4 %
	exclaim	0.001386	100.0 %
	first-name	0.001292	99.8 %
	frame	0.001480	100.0 %
	hour	0.000547	100.0 %
	last-name	0.000888	100.0 %
	minute	0.001468	100.0 %
	month	0.001806	100.0 %
	name	0.014036	98.0 %
	period	0.005075	99.4 %
	quantity	0.002295	99.8 %
	specifier	0.017490	97.4 %
	time-of-day	0.003022	100.0 %
	title	0.001695	100.0 %
	unit	0.000819	100.0 %
	Average		99.6 %
phrase features	../frame	0.000540	100.0 %
	../incl-excl	0.003584	98.2 %
	adverb	0.039457	93.8 %
	attitude	0.010799	98.5 %
	babble	0.005375	99.4 %
	conjunction	0.010376	98.8 %
	connective	0.000130	100.0 %
	degree	0.014352	96.8 %
	frame	0.110836	79.1 %
	name	0.005887	100.0 %
	specifier	0.018327	96.2 %
	type	0.001837	100.0 %
	Average		96.9 %
clause features	adverb	0.000572	100.0 %
	conjunction	0.014241	98.5 %
	degree	0.000569	100.0 %
	type	0.028168	95.5 %
	frame	0.204309	68.7 %
	Average		92.5 %
sentence features	frame	0.064294	92.5 %
	sentence-type	0.099496	79.1 %
	speech-act	0.246514	49.3 %
	Average		73.6 %

Table 4.13: Final baseline FeasPar for ESST: Test set performance on Linguistic Feature Labeler for semantic features.

Chapter 5

Cooperative Networks

In this chapter, another neural framework for finding the best feature structure is presented. The idea is to couple results from the baseline architecture, and combine them with information about how features occurred together.

This chapter first gives an analysis of the knowledge sources available (Section 5.1). Then the suggested architecture is presented and discussed in Section 5.2, followed by the presentation of results and conclusions from that.

5.1 Knowledge Sources Analysis

In various systems of different nature (cognitive and non-cognitive), the system builders combine different kinds of knowledge sources. The combination of different knowledge sources is being referred to as the *mixture of experts* principle.

This principle often has a positive impact on performance. An intuitive explanation for this is a comparison with the human decision making process: the more knowledge or facts that are available, the better the quality of the decisions.

When using this principle, the key tasks are to identify the knowledge sources, to model them, and to combine them. The latter two tasks are hard, and depend on the kind of knowledge sources.

5.1.1 Identifying Knowledge Sources

The three knowledge sources in the baseline version are:

1. **The Input Sentence Text to be Analyzed**. This is the most obvious knowledge source for any natural language understanding system.
2. **The Mapping From Input Words to Feature Pairs**. This knowledge source is provided to FeasPar as examples, and the baseline neural networks learn the mapping.

3. **The Lexicon**. The lexicon provides the word representation that the learning task needs.

However, as the results in Section 4.8 show, these knowledge sources are not sufficient to give a good performance. Therefore the following additional knowledge sources are taken into consideration:

4. **Logical Consistency** of feature pairs: It is important to understand that only certain feature combinations make sense. For the JANUS ILT, this information is already available as an ILT specification document, which is a context free grammar (see Figure 5.1 for an example rule). A program computes which feature pair combinations are consistent or not, based on the grammar.

5. **Statistical Consistency** of feature pairs: Additionally, frequencies of feature combinations can be computed based on the training material. See Table 5.1.

```
(<SIMPLE-TIME> = ((frame *simple-time)
                  (minute [NUMBER-VALUE])
                  (hour [NUMBER-VALUE])
                  (day [NUMBER-VALUE])
                  (month [NUMBER-VALUE])
                  (day-of-week [DAY-OF-WEEK])
                  (time-of-day [TIME-OF-DAY])
                  (am-pm [AM-PM])
                  (specifier [SPECIFIER])))
```

Figure 5.1: ILT specification example rule

a_1	a_2	$\#(a_1 \epsilon s \wedge a_2 \epsilon s)$	$\#(a_2 \epsilon s)$	$\frac{\#(a_1 \epsilon s \wedge a_2 \epsilon s)}{\#(a_2 \epsilon s)}$
`(frame *simple-time)`	`(day-of-week =)`	174	174	1.00
`(day-of-week =)`	`(frame *simple-time)`	174	510	0.34

Table 5.1: Statistical consistency

5.2 Architecture

A new neural network architecture called *cooperative networks* for exploiting the consistency information is suggested.

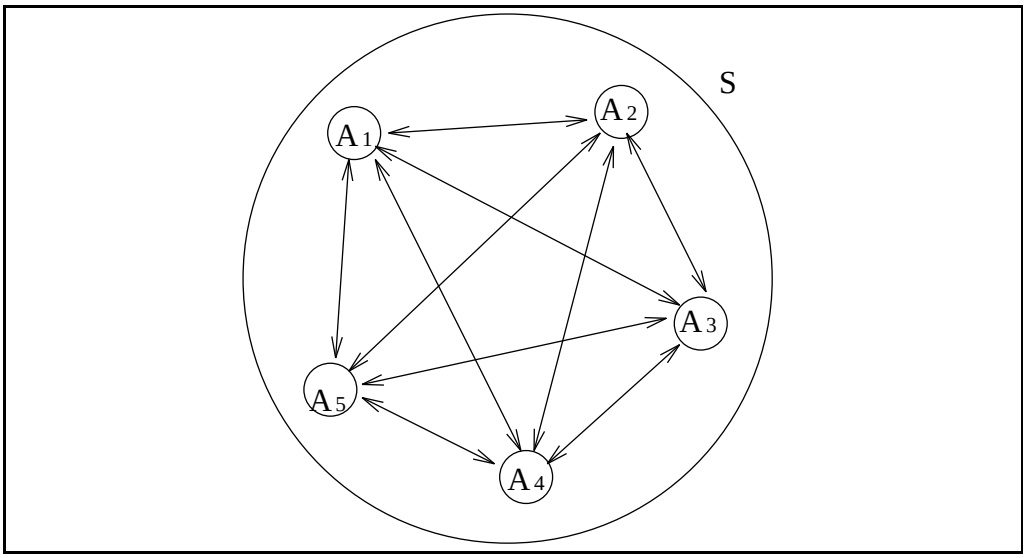

Figure 5.2: Example flat cooperative network

This architecture is explained in this section, which successively describes the various cases that must be considered. For clarity, a complete cooperative network hierarchy for a real ILT is provided in Figure 5.5, which includes all cases, and shows how they are used. The simplest case is discussed first: A flat feature structure, containing only atomic feature pairs.

5.2.1 Flat Feature Structures

The basic idea is to have a node for every possible feature pair, and connect every node with every other node (see example network in Figure 5.2 and the network S in Figure 5.5). The consistency information is expressed as weighted connections between those nodes. The nodes take as initial activation the baseline architecture output. When updating the node activations iteratively, they will influence each other. Nodes connected with positive weights, will excite each other, whereas nodes with negative weights will inhibit each other. The change of activations will converge towards zero over time. By reading out the nodes with high[1] activations, the feature structure is obtained. This intuitive explanation can be formalized. Assume a network S with the following properties:

$$A \in S \tag{5.1}$$

[1] If the value range is [d,e], then a value v is high if $v > \frac{d+e}{2}$

$$o_a := \sigma(\beta \ input_a) \tag{5.2}$$

$$w_{a,x} = \frac{\#(a \epsilon s_{any} \wedge x \epsilon s_{any})}{\#(x \epsilon s_{any})} \tag{5.3}$$

$$input_a := \sum_{x \neq a \wedge X \epsilon S} w_{a,x} o_x \tag{5.4}$$

where:

- $\#(P)$ is the number of times that event P occurred in the training data.
- a is an atomic feature pair.
- A is a node representing a.
- s is the upper most level of a feature structure.
- S a network representing s.
- s_{any} runs through all feature structures of the training set.
- x is any (atomic or complex) potential feature pair of s.
- X is a node representing x.
- $\beta = \frac{1}{T}$ is a constant. T is called the *temperature*.
- $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

Hopfield Interpretation

The cooperative network model has a strong resemblance of Hopfield network models [HKP91c]: It consists of a connected network, whose nodes get initial activations being *almost* correct. Iteration of the network calculations 5.2 and 5.4 leads to a correct feature structure. In Hopfield networks, the nodes get initial activations corresponding to an *almost* correct picture. Iteration of the network leads to a correct picture.

Both the cooperative model and the Hopfield model have two knowledge sources: The *almost* correct answer, i.e. initial activations, and knowledge about correct answers, i.e. the connection weights. In both models, the weights are calculated from training patterns. The two models differ in three aspects:

1. Cooperative networks use continuous values, whereas Hopfield networks use discrete values.

2. The Hopfield connection weights are symmetrical, whereas cooperative ones are not. The reason for the latter, is that one feature may trigger another, but not vice versa. This information would get lost if weights were symmetrical.

3. The cooperative model includes a temperature T. With high temperatures, the change in input required to flip[2] the activation, is very small. With low temperatures, it takes a lot to flip the activation. To cool off the temperature while iterating, is a good way to promote convergence, as known from Boltzman machines [HS86].

Statistical Interpretation

A cooperative network can also be given a statistical interpretation. A unit's output activation is viewed as the probability of an atomic feature value. The connection weight corresponds to the conditional probability. The total input is viewed as a sum of probability conjunctions. The recalculated output activation represents an estimate for the probability of an atomic feature value.

An *estimate* for $P(x_i)$ is calculated instead of $P(x_i)$ itself for the following reason: Feature pairs are highly dependent on each other. This means that assumptions of an independent event cannot be made. Hence, an accurate probabilistic calculation for $P(x_i)$ would lead to highly complex expressions that cannot be calculated. The probabilistic calculations would contain events where one feature depends on several other features. These events are too complex to be quantified by the information contained in a limited training data set. Therefore, the statistical model only takes into account events involving one or two feature values, and constructs an estimate based on those events.

A formal definition of the statistical interpretation is:

$$P(x_i \mid W) = o_i \tag{5.5}$$

$$P(x_i \mid x_j, W) = P(x_i \mid x_j) = \frac{1}{k_j} w_{i,j} \quad \text{(commented below)} \tag{5.6}$$

$$P(x_i x_j \mid W) = P(x_i \mid x_j, W) P(x_j \mid W) = \frac{1}{k_j} w_{i,j} o_j \tag{5.7}$$

$$input_i = \sum_{j \neq i \wedge x_j \epsilon S} k_j P(x_i x_j \mid W) \tag{5.8}$$

$$\hat{P}(x_i \mid W) := \sigma(\beta \ input_i) \tag{5.9}$$

where:

[2] To flip, here means that the output of the $\sigma(x)$ function changes from the upper end of the value scale to the lower end, or vice versa.

- W represents the input words $w_1 ... w_n$.
- $\hat{P}(x_i \mid W)$ is an *estimate* for $P(x_i \mid W)$, made by a linear combination of probabilities. The σ function ensures that the value ranges from 0 to 1.
- k_j are constants, so that the probability criteria is fulfilled: $\sum_j P(x_i \mid x_j) = \sum_{j \neq i} \frac{1}{k_j} w_{i,j} = 1$

Note that the assumption was made:

$$P(x_i \mid x_j, W) = P(x_i \mid x_j) \tag{5.10}$$

This means that:

$$P(x_i \mid x_j, W) = P(x_i \mid x_j) = \frac{1}{k_j} w_{i,j} = \frac{1}{k_j} \frac{\#(i \epsilon s_{any} \wedge j \epsilon s_{any})}{\#(j \epsilon s_{any})} \tag{5.11}$$

The assumption is introduced for simplicity, since there would not be enough data available to calculate the frequencies in respect to W, which would be the expression: $\frac{\#(x_i \epsilon s_{any} \wedge x_j \epsilon s_{any} \wedge W)}{\#(x_j \epsilon s_{any} \wedge W)}$. The Hidden-Markov-Model [Rab90] assumption, where only the last state of a sequence of states is considered, is a similar assumption.

5.2.2 Nested Feature Structures

So far, feature structures consisting only of atomic feature pairs have been discussed. However, since a feature structure contains substructures, these structures must all be represented somehow. A simple extension within the framework is to let every structure and substructure be represented by a network, and let a complex feature be represented by a node that is a member of both networks (see example network in Figure 5.3 and the networks S and C_1 in Figure 5.5). The node takes activations from both nets as input, and contributes to the input of all other nodes of both nets. Formally, this amounts to the following extensions:

Assume:

1. A complex feature pair $(f \ c_j)$, with feature f and complex value c_j.
2. $(f \ c_j)$ is part of feature structure s.
3. Consequently, c_j is a substructure of s (follows from 1. and 2.).

Then let:

4. a node $Q_{(f \ c_j)}$ represent $(f \ c_j)$.

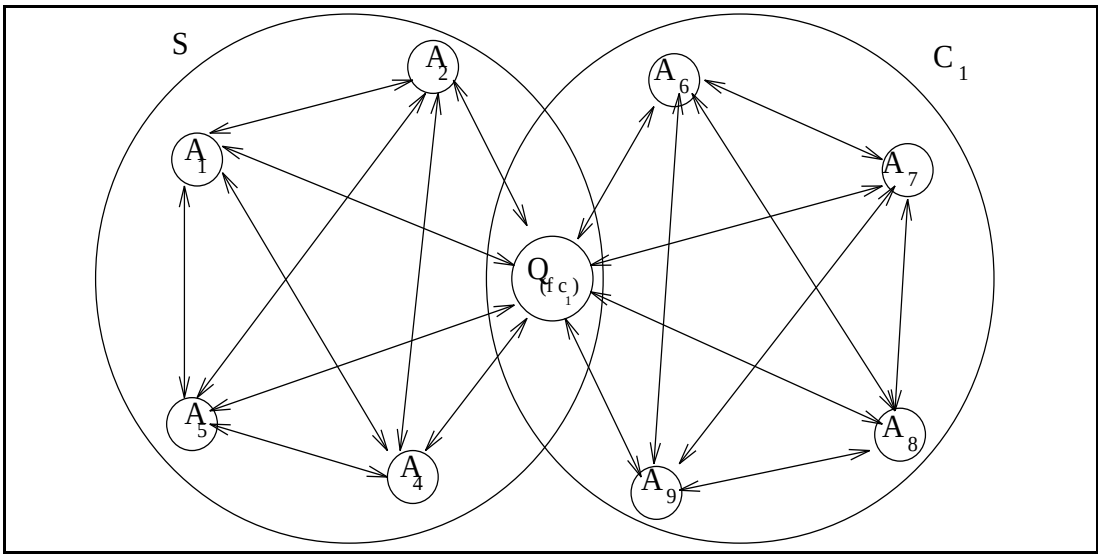

Figure 5.3: Example complex value cooperative network

5. a network S represent s.

6. a network C_j represent c_j. $Q_{(f\ c_j)}$ is called the *parent* node of C_j.

7. $Q_{(f\ c_j)}\ \epsilon\ S \wedge Q_{(f\ c_j)}\ \epsilon\ C_j$.

In total, there is an interaction of two effects: On one side, in each network the nodes will adjust to each other. This means the complex feature node $Q_{(f\ c_j)}$ will get a high or low activation, i.e. the other nodes in S will 'tell' the complex feature node $Q_{(f\ c_j)}$ whether it 'fits in' in S or not. Further, the activation o_c then influences the nodes in C_j, so that if o_c is high, node representing relevant features in C_j gets high, and vice versa. On the other side, the nodes in C_j will influence o_c, i.e. they will 'tell' the complex feature node $Q_{(f\ c_j)}$ whether it fits as parent for C_j. Formally, this means that the cooperative network model is extended with the following formulas:

$$input_{(f\ c_j)} := \sum_{x \neq (f\ c_j) \wedge X \epsilon S} w_{(f\ c_j),x} o_x + \sum_{x \neq (f\ c_j) \wedge X \epsilon C} w'_{(f\ c_j),x} o_x \qquad (5.12)$$

$$w_{(f\ c_j),x} = \frac{\#((f\ c_{any}) \epsilon s_{any} \wedge x \epsilon s_{any})}{\#(x \epsilon s_{any})} \qquad (5.13)$$

$$w'_{(f\ c_j),x} = \frac{\#(x \epsilon c_{any} \wedge (f\ c_{any}) \epsilon s_{any})}{\#(x \epsilon s_{any})} \qquad (5.14)$$

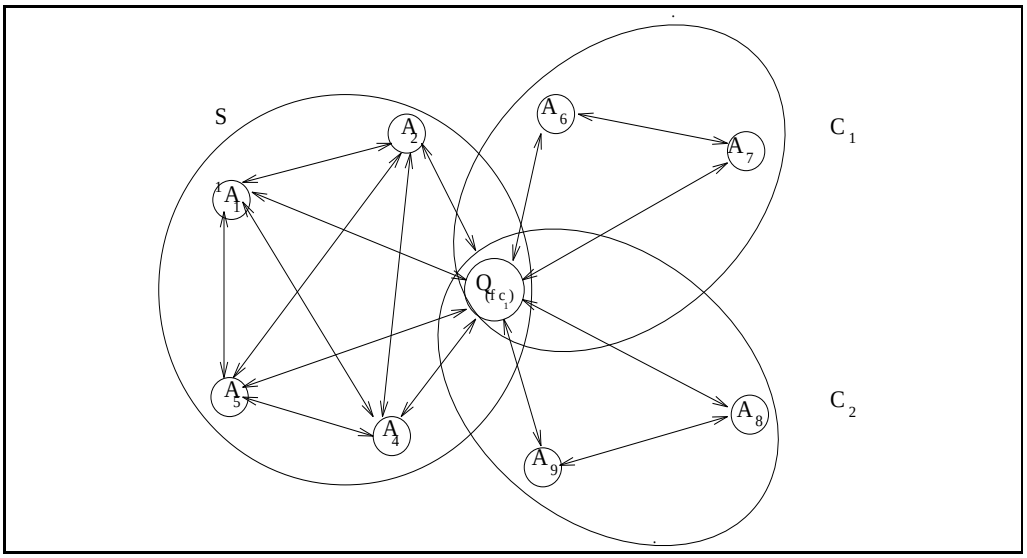

Figure 5.4: Example multiple complex value cooperative network

5.2.3 Multiple Feature Values

Features normally take one value. However, sometimes, they take two or more. These values may be either all atomic or all complex.[3] This section shows how to fit multiple featur values into the frame work.

multiple atomic values are simply handled by letting e.g. the feature pair $(f_1\ a_{1,1})$ be represented by another node than $(f_1\ a_{1,2})$. All equations are then valid also for these two nodes.

multiple complex values of the feature f are represented by one node Q_c in network S, and one network C_i per complex value c_i (see example network in Figure 5.4 and the networks C_{11} and C_{12} in Figure 5.5). Node $Q_{(f\ c_j)}$ is member of S and all C_i. To avoid dominance from all C_i versus S in influence on o_c, the average, rather than the sum, of the impacts from all C_i are used:

$$input_{(f\ c)} := \sum_{x \neq c \wedge X \epsilon S} w_{c,x} o_x + \frac{1}{n}\left(\sum_{x \neq c \wedge X \epsilon C_1} w'_{c,x} o_x + .. + \sum_{x \neq c \wedge X \epsilon C_n} w'_{c,x} o_x\right) \tag{5.15}$$

[3]Both types of values never occur together.

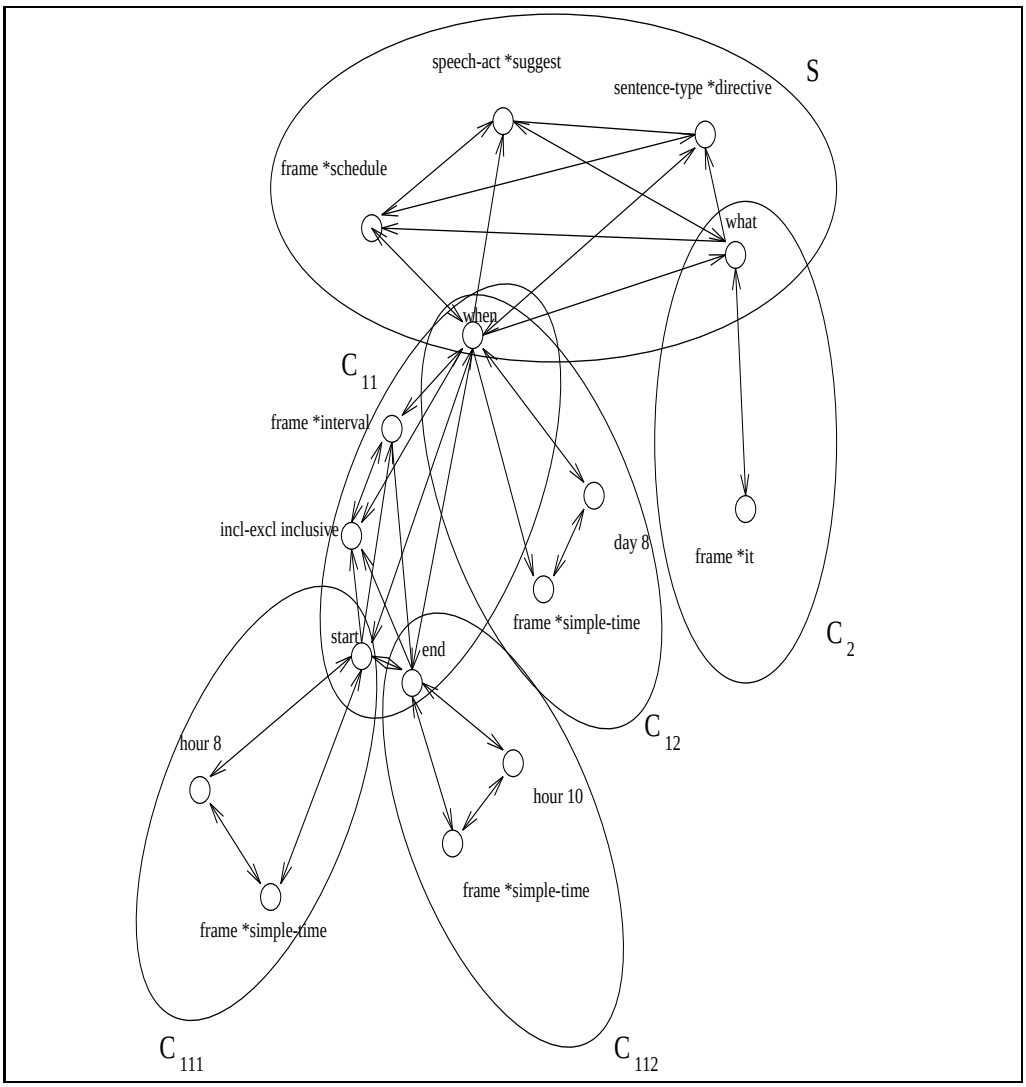

Figure 5.5: Real example cooperative network for the ILT in Figure 4.13

5.2.4 Initialization Values

Another topic is how to initialize the o_a and o_c values. In principle, they should be initialized with the results from the baseline parser, as presented in the last chapter. The baseline architecture yields activations for atomic feature pairs (the Linguistic Feature Labeler networks) and for chunk paths (the Chunk Path Finder). The suggested way to combine these activations is to multiply the activations of each path element and the atomic feature value activation. The product is normalized with respect to the number of factors in the product by taking the nth root, because various path segment combinations may lead to the same path.

Formally, the cooperative model is extended with the formulas:

Initial activation for atomic feature pair node A in network N:

$$o_a^{init} := \sqrt[m]{\underbrace{o_{path_{0,0}}^{baseline} .. o_{path_{0,k_0}}^{baseline} .. o_{path_{l,0}}^{baseline} .. o_{path_{l,k_l}} o_a^{baseline}}_{m \text{ factors}}} \tag{5.16}$$

Initial activation for complex feature pair node C in network N:

$$o_c^{init} := \sqrt[m]{\underbrace{o_{path_{0,0}}^{baseline} .. o_{path_{0,k_0}}^{baseline} .. o_{path_{l,0}}^{baseline} .. o_{path_{l,k_l}}}_{m \text{ factors}}} \tag{5.17}$$

where:

- $o_a^{baseline}$, is the neural network output activation for the atomic feature value v of the network for the feature f in the Linguistic Feature Labeler, i.e the feature pair a = (f v).
- $o_{path_{i,j}}^{baseline}$ are the activations for the chunk path segments values. $path_{0,0}..path_{k,l_k}$ together make up the path for N referred to the upper most network.
- N is a network at any level, including the upper most network.

5.3 Experiments

The first implemented version of the cooperative network model has the following disadvantages:

1. All units are asymptotically drawn towards zero. The reason for this is that many activations and weights have values in the lower half of their range. The range is initially $[0, 1]$. Experiments with other value ranges showed that the most promising is $[-1, 1]$ for both activations and weights.

2. Some nodes representing the same feature pair, beloging to different networks, due to different paths, become activations being all high. Since the interpretation of the paths is that only *one* of them is correct, only one of these feature pair nodes should get high activation. Therefore, inhibitory global constraint weights are introduced.

3. When the activations get stable in sensible value ranges, it turns out that activations stabilize at values very far from the initial values, i.e. that the networks stabilize into feature structures that have very little to do with the feature structures from the baseline version. Therefore, in later implementations, o_x^{init} may also impact the activation calculation in later iterations.

PM 1:

System	precision	normal
LR	69.5%	51.6%
FeasPar baseline	58.3%	33.8%
FeasPar cooperative networks, experiment 0	23.3%	-26%
FeasPar cooperative networks, experiment 39	66.8%	-42%
FeasPar cooperative networks, experiment 44	43.2%	-81%

Explanations:

- 'normal' is the measure defined in Section 3.4.1.
- 'precision' is the measure as defined in Section 3.4.1, but allows extra features and/or feature values in the output, without counting these as wrong. 'Precision' is a weaker criterion than 'normal'.

Parameter settings:

#	z_o	z_w	w_{init}	w_o	o_{min}	w_{min}	w_{incons}	η_{path}	Δ_{stop}	T_{start}	T_{decay}
0	-1.0	-1.0	1.0	1.0	10^{-5}	0.05	-10^2	10	10^{-5}	1.0	0.9
39	-1.0	-1.0	1.0	0.0	10^{-5}	0.05	-10^4	100	10^{-2}	1.0	0.95
44	-1.0	-1.0	1.0	0.0	10^{-5}	0.05	-10^2	100	10^{-2}	1.0	0.95

Figure 5.6: Cooperative networks results

Therefore, the second implemented version additionally contains these parameters:

- z_o, output zero point. The output value range is $[z_o, 1]$

- z_w, weight zero point. The weight value range is $[z_w, 1]$
- w_{init}, init weighting, and w_o, output weighting. Equation 5.2 is replaced by:

$$o_a := \sigma(\beta(w_{init} o_a^{init} + w_o input_a)) \tag{5.18}$$

- o_{min}: if $o_a < o_{min}$ then the node Q_a is not created in the network to represent A.
- w_{min}: if X_i and X_j are consistent, but $w_{i,j} < w_{min}$ then $w_{i,j} := w_{min}$
- w_{incons}, inconsistent weight. If X_i and X_j are inconsistent, then $w_{i,j} := w_{j,i} := w_{incons}$
- η_{path}, path factor. In order to strengthen the significance of structure, the output of a node, Q_c for complex feature value (see 5.2.2), is multiplied by η_{path} before being used as input activation in the sums in Equations 5.4 and 5.12.
- Δ_{stop}. Stop criteria for the iterations: If the sum of output changes is less than Δ_{stop}, then these networks are assumed to have reached stability.
- T_{start}. Starting value for T.
- T_{decay} Decay factor for T: $T_{t+1} = T_t T_{decay}$

Most parameter combinations cause one or more problems: most units get saturated at the top or bottom of the value range; or the activations end up oscillating instead of stabilizing. The most successful parameter combinations[4] and their experimental results on the entire evaluation set are shown in Table 5.6.

5.4 Summary

The results clearly state that cooperative networks do not give the desired performance compared to the baseline version, given the ESST task and data. However, since the method is theoretically sound, and has strong parallels to other successful models, e.g. the Hopfield model, it may provide good results with other tasks. In the opinion of the author, the cooperative network model requires more data to work well, like other statistically motivated models do [BCP+90, BLM91a, BPP+91].

Though, for tasks with little training data as in ESST, the cooperative networks do not work well, and other methods must be used for improving the performance. A method more successful for ESST is presented in the next chapter.

[4] In total 45 combinations were tested.

Chapter 6

Consistency Checking Search

In the previous chapter, the knowledge analysis shows that there are more knowledge sources available than those that are used in the baseline system. However, it is also shown that cooperative networks do not provide the desired performance increase. This chapter presents another method for adding the extra knowledge source that leads to a substantial performance increase.

6.1 Knowledge Sources Analysis

As already stated in Section 5.1, the baseline parser of chapter 4 should be enhanced with consistency information. The complete parse depends on many neural networks. Most networks have a certain error rate; only a few networks are perfect. When building complete feature structures, these network errors multiply up, resulting in not only that many feature structures are erroneous, but also inconsistent and making no sense. A search algorithm compensates for this. It is based on two main information sources: first, probabilities that originate from the network output activations; second, a formal feature structure specification, stating what combination of feature pairs is consistent. This specification is already available as an interlingua specification document (see Section 5.1).

6.1.1 Global Constraints

Additionally, a few other ILT constraints must be considered. that are not modeled in the interlingua specification document. They are called global[1] constraints, and include three types:

[1] The term *global* is used for reasons explained in Section 6.3.

1. **Frame Constraint**: An ILT is a feature structure, where at each branch the feature `frame` has one and only one value.

2. **Up-Features** (see Section 4.3) as e.g. `../incl-excl` may not appear at the top most branch, because no parent branch is available.

3. **Compulsory Constraints:** Not only a feature pair F_1 *may* appear with another feature pair F_2, but that F_1 *must* appear with F_2, i.e. in some sense, F_1 triggers F_2. An example is `(frame *simple-time)` and `(day-of-week =)`. The frequencies in the right-most column of Table 5.1 suggest that `(day-of-week =)` triggers `(frame *simple-time)`.

6.2 Architecture

6.2.1 Search Task

In combining the network output and the constraints, the search finds the feature structure with the highest probability, under the given constraints being consistent. The outputs of each neural network are normalized to give a probabilistic interpretation. Then they are sorted by probability. They can now be viewed as an N-best list. Hence, the search input is one N-best list per network. To combine these N-best lists hierarchically to build an N-best list of feature structures, forms the search task.

6.2.2 Search Complexity Precautions

The ESST baseline version of FeasPar had 37 Linguistic Feature Labeler Networks and 4 Chunk Path networks. Each network has up to 15 different output values. It is crucial to keep complexity and search times low. Therefore, the following principles and constructs are applied:

Hierarchy of Feature Structure Fragments: A feature structure is assembled using partial feature structures. These are called *fragments*. The hierarchy corresponds to the chuck hierarchy and in what sequence the fragments are put together to form a complete feature structure (see the algorithm in Figure 4.4).

Agendas: Agendas (one per fragment) are used to direct the search, so that always the most probable of the unexamined combination is examined first.

Lazy Evaluation: The Lazy Evaluation delays the expensive calculations of fragments and agenda as long as possible. This is extremely important to reduce search time.

6.3 Search Principles

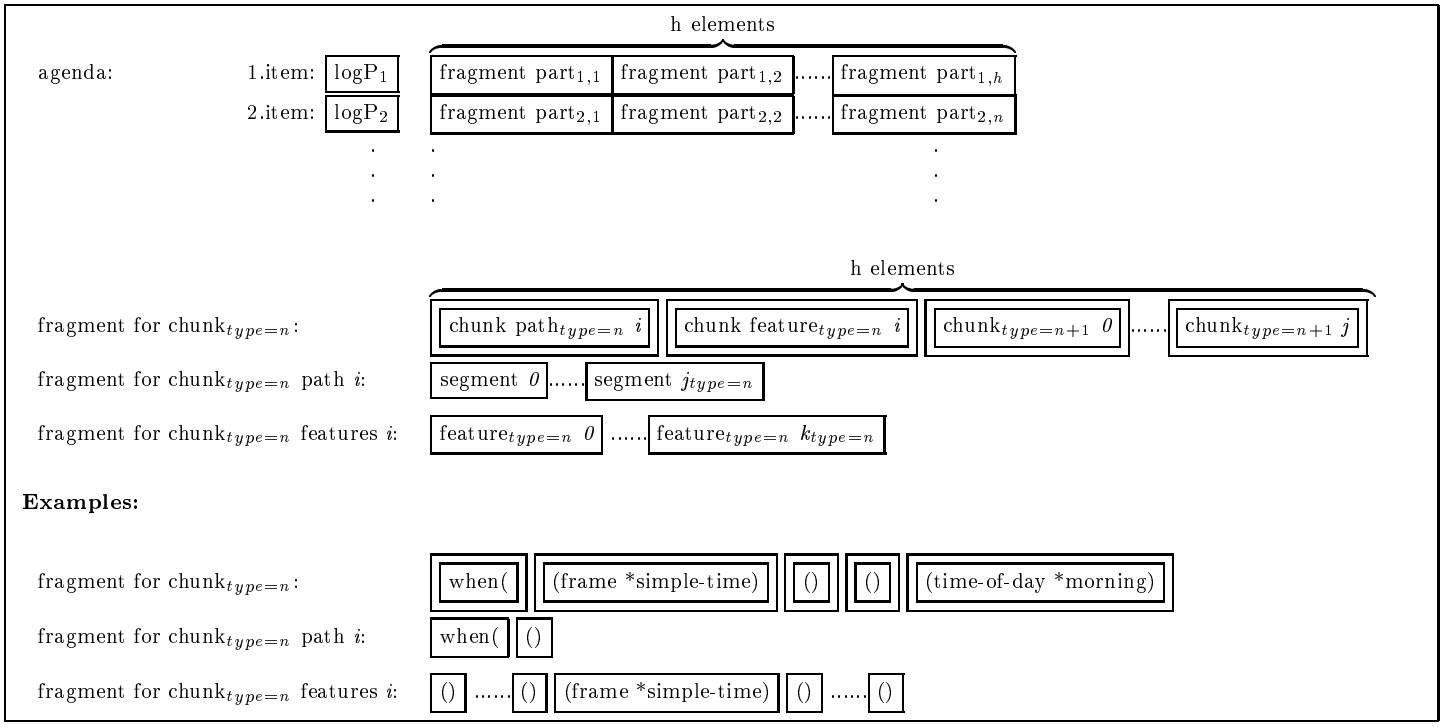

Figure 6.1: **Agenda and fragments.** Double framed fragment parts indicates another fragment. Single framed fragment parts indicates neural network output.

When building a feature structure, the search uses structures as shown in Figure 6.1: For the partial feature structure of every chunk, it defines an N-best list of fragments. The fragment parts correspond to chunk path, chunk features, and subchunks.

Example: The fragment example for $\text{chunk}_{type=n}$ in Figure 6.1 corresponds to the chunk:

([**when**]((frame *special-time))
 ([] in)
 ([] the)
 ([]((time-of-day =morning)) morning))

Building a fragment is an expensive operation. In order to build fragments as few times as possible, an agenda is used in parallel to each fragment list.

Agenda calculations are much cheaper than fragment calculations. The agenda and fragment interact as follows:

The agenda keeps hold of possible fragment configurations. It is sorted by log probability. Upon a request for a new fragment, the next configuration is fetched from the agenda, and the fragment is built. If during fragment building, an inconsistency is detected, the building operation is abandoned, and the next element on the agenda is used as configuration. Then a new fragment is built. This continues until a complete consistent fragment has been built. This fragment is then stored in the N-best list of its chunk, and returned.

During building, fragment parts must be fetched. These are mostly fragments themselves (e.g. in Figure 6.1 fragment for chunk $\text{path}_{type=n}\ i$). If this fragment part has already been calculated during the search, it is already available in the N-best list. If not, a request for a new fragment is made.

The agenda itself is expanded as little as possible: When a new agenda item has been accessed, those candidate agenda items that may follow immediately are inserted in the agenda. This avoids an combinatorial explosion, but ensures that no configuration is left out, or tried too late, with respect to logP (the logarithmic probability).

The consistent constraints mentioned above, are derived as follows: A feature structure formalism contains rules that express in which context what feature pairs may appear. Prior to the parsing process, the program statically calculates for every combination of two feature pairs, if the two feature pairs may occur together or not. This information is consulted during fragment building, as mentioned above.

Global constraints (see Section 6.1.1) can only be tested on the complete feature structure. When the search returns a complete feature structure for the upper most chunk, the global constraints are tested on the feature structure. If a test fails, the search is continued, until a complete feature structure satisfying all global constraints have been found.

Even if all possible care is taken to speed up the search, the worst-case search is too long. To prevent this, the search is broken off at a certain depth, and the search is repeated, this time allowing *one* inconsistency. If this search gets too deep, two inconsistencies are allowed, and so on.

6.3.1 Search Implementation

In the following, the search algorithm will be explained in more detail. Its core consists of three interacting procedures: (See also Figure 6.2 - 6.5.)

get_fragment: This procedure produces the feature structure fragment corresponding to a chunk. Calculation is only started if the fragment has not previously been computed, otherwise the corresponding item from the N-best list is returned. Before the fragment is calculated, all immediate succeeding agenda elements are calculated and inserted in the agenda. A

check is also made to the search depth to prevent search time from getting too long.

get_subfragment: This procedure is called to increase the agenda: The call (in get_fragment) is made in a loop. In this loop, for the subchunks of the currently examined chunk, each subchunk is examined one element deeper in its N-best list, the so-called *new list element*. The notion of subchunk includes chunk paths, chunk features, as well as three subchunks. get_subfragment therefore has two purposes. First, the feature structure fragment corresponding to a subchunk is fetched. Second, the agenda is expanded by the new list element. This ensures two things: one, that the agenda is expanded as late as possible, e.g. only one step in each direction (subchunk) from the current agenda position; and two, that its not generated too late, after it should have been evaluated. The latter is true, because any agenda element N is generated when a agenda element O is calculated, where O differs from N by one list element, and therefore O is more probable than N, and N should therefore be evaluated after O.

get_subfragment_lazy: Perform a lazy evaluation of the subfragment, by making a look-up in the respective N-best list.

6.4 Improvements

The following improvements are added to FeasPar in order to gain performance:

Rescoring and N-best: Since the Consistency Checking Search in principle can deliver an N-best list of resulting ILTs, the potential for rescoring (i.e. not using neural network generated probabilities, but some other measure for choosing the number one ILT candidate) was examined. In an initial experiment, the PM1 calculation program is changed, so that it calculates PM1 for the best 20 ILTs for every sentence, and picks the one with fewest errors for summing up. This gives an error reduction to about the half[2]. The only remaining (non-trivial) problem is to find a rescoring that would pick a better number one alternative than the neural network probability does. A few different scores have been tried without success. Therefore, the rescoring and N-best principle is not used in the final Consistency Checking Search version.

Allowing Multiple Equal Feature Pairs: Occasionally, when building a fragment during a search, more than one subfragment contains the same

[2] In that early Consistency Checking Search version, PM1 using first-best, is 54.4 %, and using N-best 76.9 %

```
typedef struct CHUNK_STRUCT{
  struct FRAGMENT_LIST_STRUCT *attr_fragment_list;
  struct FRAGMENT_LIST_STRUCT *path_fragment_list;
  struct CHUNK_STRUCT *subchunks;
  int n_subchunks; /* the *total* number of subchunks is   */
                   /* 'n_subchunks+2', since               */
                   /* number of 'subchunks' is n_subchunks */
  struct FRAGMENT_LIST_STRUCT *fragment_list;
} CHUNK;

typedef struct FRAGMENT_LIST_STRUCT{
  struct FRAGMENT_STRUCT *fragments;
  int n_fragments;
  struct AGENDA_STRUCT *top; /* agenda top */
  struct AGENDA_STRUCT *calc;  /* currently calculated element */
} FRAGMENT_LIST;

typedef struct AGENDA_STRUCT{
  int *subfragment_n_best;
  double logP;
} AGENDA;

typedef struct FRAGMENT_STRUCT{
  double logP;
  struct FEATURE_PAIR_STRUCT *feature_pairs;
} FRAGMENT;
```

Figure 6.2: Search algorithm: data types. See the following figures for usage. Only relevant information is shown for clarity.

feature pair, i.e. more than one chunk is responsible for adding a particular feature pair at a particular feature structure branch. This occurs even if it is not supposed to happen, according to the principles of the hand modeled alignment. The earlier Consistency Checker Search does not accept this, and requires that one and only one instance of a feature pair is produced for a particular branch. A later version allows multiple feature pair instances.

Sloppy Lexical Feature Value: As described in Section 4.3 and Section 4.3.1, FeasPar uses lexical feature values. These are collected from the

```
FRAGMENT *get_fragment(CHUNK *chunk, int n_best)
{
  FRAGMENT **fragment;
  int i, j;
  AGENDA *next_calc;

     /* if fragment has already been calculated,
        get old result by looking up in table */
  if (n_best <= chunk->fragment_list->n_fragments-1){
    (*fragment) = &(chunk->fragment_list->fragments[n_best]);
    return(*fragment);  }
  else
    while (1){
      if (chunk->fragment_list->top != NULL){
        for (i=0; i<chunk->n_subchunks+2; i++){
          j = chunk->fragment_list->calc->subfragment_n_best[i];
            /* get subfragment and produce new agenda element,
               and extend agenda   */
          get_subfragment(chunk,i,j+1);        }
          /* get next agenda element */
        next_calc = next_agenda_e(chunk->fragment_list->calc);
        if (next_calc == NULL) return(NULL);
        chunk->fragment_list->calc = next_calc;
      }
      if (break_search(chunk->fragment_list)){
        printf("QUIT DUE TO DEEP SEARCH \n");
        return(NULL); }
         /* evaluate (eager) next fragment, and return if legal */
      if ((calculate_fragment(chunk,fragment)) &&
           ((check_global_constraints(fragment)))){
          save_fragment(fragment,chunk->fragment_list);
          return(*fragment);  }
      }
  }
  return(NULL);
}
```

Figure 6.3: Search algorithm: get_fragment. See text for further explanation.

```
FRAGMENT *get_subfragment_lazy(CHUNK *chunk, int i, int j)
{
  FRAGMENT_LIST *fragment_list;

  if (i == chunk->n_subchunks)
    fragment_list = chunk->attr_fragment_list;
  else if (i == chunk->n_subchunks+1)
    fragment_list = chunk->path_fragment_list;
  else
    fragment_list = chunk->subchunks[i].fragment_list;

  if (fragment_list->n_fragments <= j) return(NULL);
  else return(&(fragment_list->fragments[j]));

}
```

Figure 6.4: Search algorithm: get_fragment_lazy. See text for further explanation.

training data and stored in the lexicon. However, due to incompleteness or speech recognizer errors, a situation may arise, where a natural language chunk is not being stored in the lexical feature value lookup table. In many cases however, a similar chunk may be present, and could be used. An example will clarify this:

Assume that in the lexical feature value table, only the phrase chunk "out of town" is stored, with the lexical feature value `*out-of-town` for the `frame` feature. Now, the phrase chunk "out of the town" causes the `frame` feature network output value for lexical feature values to have the value $act = 0.85$. However, since "out of the town" is not stored in the lexical feature value lookup table, instead of failing, the sloppy mechanism compares with table entries, computes the difference, $diff$ [3] as well as storing the table value, `*out-of-town`. A new activation value $act_{sloppy} = act * k^{diff}$ is calculated and used instead of the original act in further processing. ($0 < k < 1$ is an empirical constant. 0.9 works well.) Then, the alternatives are sorted by increasing $diff$, so that in the example, the top most alternative value for `frame` is `*out-of-town` with $act_{sloppy} =$

[3]The difference measure, $diff$, is the same as used when calculating word accuracy in a speech recognizer, i.e. the minimal number of insertions, deletions and substitutions steps needed to change one string to the other.

```
void get_subfragment(CHUNK *chunk, int i, int j)
{
  FRAGMENT *fragment,
           *old_fragment;
  AGENDA *new_agenda;

  if (i == chunk->n_subchunks)
    fragment = get_attr_fragment(chunk,j);
  else if (i == chunk->n_subchunks+1)
    fragment = get_path_fragment(chunk,j);
  else
    fragment = get_fragment(&chunk->subchunks[i],j);

  if (fragment == NULL) return;

  new_agenda = copy_agenda(chunk->fragment_list->calc,
                           chunk->n_subchunks+2);
  old_fragment =
          get_subfragment_lazy(chunk,i,
                               new_agenda->subfragment_n_best[i]);
  new_agenda->logP = new_agenda->logP - old_fragment->logP
                     + fragment->logP;
  new_agenda->subfragment_n_best[i] = j;
  insert_in_agenda(new_agenda,chunk->fragment_list->top,
                   chunk->n_subchunks+2);
}
```

Figure 6.5: Search algorithm: get_subfragment. See text for further explanation.

$$0.85 * 0.9 = 0.765$$

Remodeling of (frame *busy) and (frame *free): Initial experiments with the performance measure 2 shows that a high error rate is due to confusions between (`frame *busy`) and (`frame *free`). These were during manual labeling of the training data mostly aligned with the clause chunks. The corresponding network has quite a high error rate. However, since these features can easily be remodeled and aligned with one or more phrase chunks, this was done. The retraining then only involved two networks (for the `frame` feature for clause and phrase chunks).

Constraint Relaxation: One important problem with the search algorithm is that sometimes (1 % to 3 % of the analyses), the search takes too long, [4] and therefore has to be broken off. This is due to the worst case scenario, where all combinations must be searched to find a consistent configuration.

To escape from infinite searches, is the purpose of the following break strategy: If a fragment N-best list exceeds a fixed large number, e.g. 5000, then the search is stopped, and an empty ILT is returned as a parse result. However, since it is better to get a suboptimal analysis than no analysis at all, a constraint relaxation mechanism is added: If a search is broken off, then a new search is made, where one constraint may be relaxed. If this does not give any parse result, then two inconsistencies are allowed etc. (A maximum number of inconsistencies, e.g. 7, is there to prevent infiniteness.)

6.5 Evaluation

FeasPar is compared with a hand modeled GLR* parser. The hand modeling effort for FeasPar is 2 weeks. The hand modeling effort for the GLR* parser is 4 months. All performance measures are described in Section 3.4. The parsers are evaluated with the evaluation set (Set 3). Results are shown in Figure 6.1.

	FeasPar (with Search)	FeasPar (without Search)	GLR* parser (4 months)
PM1 - T	71.8 %	33.8 %	51.6 %

	FeasPar (with Search)	GLR* Parser (4 months)
PM1 - T	71.8 %	51.6 %
PM1 - S	52.3 %	30.3 %
PM2E - T	74 %	63 %
PM2E - S	49 %	28 %
PM3G - T	49 %	42 %
PM2G - S	36 %	17 %

Table 6.1: Comparing FeasPar with a GLR* parser hand modeled for 4 months (Evaluation set (Set 3), S=speech data, T=transcribed data).

As one can see, FeasPar with Consistency Checking Search is better than the GLR* parser in all six comparison performance measurements that are made.

[4] In normal cases (97 % to 99 % of the analyses), the search takes 1 to 3 seconds. In the remaining few cases, the search can run for 10 minutes without completing.

6.6 Final Evaluation

At last, a final evaluation on a new, final evaluation set (Set 4) is made. The purpose is threefold:

1. To make sure that search development has not been tuned towards the first evaluation set (Set 3).
2. Run a comparison with the newest GLR* grammar, that has been developed for 2 years[5].
3. Examine if insertion of new words into the lexicon reduces performance. Set 4 contains 60 words not covered by the Sets 1-3. These words are added manually as new words to the lexicon before performing the final evaluation. Each new word is defined by copying and modifying features of similar words. The process of extending the lexicon with the 60 words takes approximately 4 hours.

	FeasPar with Search		GLR* Parser - 2 years	
	independent grading	(my grading)	independent grading	(my grading)
PM2E - T	75.1 %	(75.7 %)	78.6 %	(78.6 %)
PM2E - S	60.5 %	(63.5 %)	60.8 %	(61.5 %)

Table 6.2: Comparing FeasPar (old ILT) with a GLR* parser (new ILT) developed over 2 years. (Final evaluation set (Set 4), S=speech data, T=transcribed data)

6.6.1 Results

All output was graded by an independent person, a student whose native language is English and not involved in any parser research or development, and the author. Grading results are shown in Figure 6.2. By referring back to the purposes of this comparison, one sees that:

1. When comparing Table 6.2 with 6.1, one can see that for FeasPar, PM2E-T is practically the same: 75.1 % (75.7 %) vs. 74 %. This shows that the development of the search is not based on information contained in the evaluation set (Set 3). Further, one can see that for FeasPar, PM2E-S is better in the latter comparison. This is due to the ESST speech recognizer improvement from March 94 to September 95.

[5] To be totally correct: almost 2 years, that is 22 months.

2. Further, from Figure 6.2 one sees that for speech data, which is the situation for being used in an actual speech-to-speech translation system, FeasPar and the GLR* parser have practically the same performance.

3. Adding the new words caused no reduction of performance (refer to point 1 above). This shows that the system generalizes well also in respect to new words.

6.6.2 Comparison Comments

For completeness, this subsection explains some considerations that had to be made for running the final comparison.

FeasPar is trained with the March 94 ILT version (in the following called the *old ILT*), whereas the newest GLR* grammar uses the September 95 ILT version (in the following called the *new ILT*), being changed over 50 times (!) since March 94. [6] Therefore, some steps are taken to avoid comparisons that would be biased towards FeasPar:

Not Correcting for GLR*'s ILT Performance Advantage: Because the GLR* grammar had a more sophisticated ILT, its translation performance (English-to-English) would of course be better than if it would have used the old ILT. This *ILT performance advantage* is estimated by the GLR* author to be slightly more than 10 % [Lav96a]. Note that the numbers in Table 6.2 are **not** corrected due to this, in order not to favor FeasPar.

German Generator: The German generator was only developed until the March 94 evaluation. This means that it knows all constructs of the old ILT, but has problems with the September 95 ILT. This means that it would generate well from FeasPar's output, but not so well from the 2 year GLR* parser's output (new ILT), meaning a comparison would be in favor of FeasPar. Therefore, no English-to-German translations are run, and hence, the performance measure 2G is not made.

English Generator: The English generator knows the new ILT, so it can cope with both the output of the 2 year GLR* parser (new ILT), and the output of FeasPar (old ILT). The English generator is still able to cope with the old ILT, because during generator development, constructs were mainly added, but not removed. If certain constructs were removed, it could mean a performance decrease for FeasPar, but by no means a performance increase.

Sentence Break Finding: In the GLR* system for parsing speech data, symbolic preprocessing which looks for pauses and silences, splits the utterances into sentences. Since FeasPar is trained with sentences, the utter-

[6] Due to time constraints, FeasPar was not retrained on the newest ILT version.

ances are run through this preprocessing to divide them into sentences before presenting them to FeasPar.

Further, the final evaluation set does not include ILTs, only natural language. Hence, performance measure 1 cannot not be made. As a conclusion, only performance measure 2E can be made.

The GLR* parser output was produced in an internal GLR* project evaluation at Carnegie Mellon University July 15 1995.

Due to the skip part of the GLR* parser system, the speech data translations are several times split into smaller natural language pieces. In order to favor the GLR* parser, these pieces are graded individually, so that e.g. a sentence split up into 2 pieces, where 1 pieces are acceptable and 1 piece are non-acceptable, this is counted as 1 acceptable and 1 non-acceptable (piece) translations, and not as 1 (sentence) non-acceptable. Therefore, for speech data, 374 pieces are graded for the GLR* parser, and only 348 for FeasPar.

Chapter 7

Evaluation

This chapter contains an overall evaluation of this thesis. First, a comparison with other approaches is described. The usefulness for various tasks and the ease for non-experts to build a parser are then explained. Finally, other advantages are discussed.

7.1 Comparison with Other Approaches

This section will compare FeasPar with hand modeled grammars and connectionist systems. For clarity, the comparison only includes systems evaluated on similar tasks as FeasPar.

7.1.1 Comparison with Hand Modeled Grammars

Symbolic parsers with hand modeled grammars have the advantage of high performance. Further, many symbolic parsers yield a parse output formalism allowing fine-grained descriptions of the analysis.

The GLR* parser, based on Tomita's efficient parsing algorithm, uses an unification engine and a hand model parsing grammar. Several GLR grammars, syntactic and semantic, both for text and speech, have been written over the years. Much work has been invested to successfully improve its robustness for usage with speech and especially spontaneous speech. The GLR* parser possesses a very powerful output formalism, feature structures, which are applied (in variants) in all unification based parsers and have been shown to be expressive enough to model all types of linguistic information, ranging from phonetics to discourse modeling.

For the GLR* grammars for the English Spontaneous Speech Task (ESST), two samples of the grammar were selected: The first after 4 months of development, and the second after two years of development. In order to test not only

parse performance, but also translation performance, a GLR* generation grammar for English and one for German are added to get a complete translation system.

Compared with the GLR* grammar developed for 4 months, FeasPar has a better performance both in the parse performance (PM1) as well as in performance for acceptable translations into English (PM2E) and German (PM2G). This applies both for transcribed and speech input (see Figure 6.1), using the evaluation set (Set 3, see Section 3.3 on page 25).

On the unseen, final evaluation set (Set 4), FeasPar has a similar performance (60.5 % versus 60.8 %) as the GLR* grammar developed for 2 years, when measuring the performance for acceptable translations into English (PM2E) with speech input. For transcribed input, i.e. input not processed by the speech recognizer, the GLR* grammar performs slightly better than FeasPar (78.6 % versus 75.1 %)[1].

A performance comparison for German is not provided, because it would be positively biased towards FeasPar for technical reasons. Since the final evaluation set used for performance tests, did not include correct parser output (ILTs), it is also not possible to measure parse performance.

The RTN based Phoenix parser, whose grammar was developed over 9 months, has a similar performance. However, FeasPar has a richer parse formalism than Phoenix, since RTNs do not include attributes.

Summing up, FeasPar has a better or equal performance as hand modeled grammars that have been developed for months or years, whereas FeasPar itself only needs 2 weeks of hand modeled information.

7.1.2 Comparison with Connectionist Parsers

The main advantages of connectionist parsers are robustness and learning capabilities. PARSEC is the parser that is closest related to FeasPar. PARSEC was evaluated with read speech, whereas FeasPar is evaluated with spontaneous speech. A direct performance comparison is therefore not possible. PARSEC's output formalism contains three (architecturally fixed) levels with one label per level, and no attributes (see example in Figure 7.1). FeasPar's output formalism has no architecturally fixed maximum levels and allows several (not architecturally fixed) features, with atomic or complex values (see example in Figure 7.2). The architectural, connectionist and learning principles that are successfully applied in PARSEC, proved to be insufficient for the parsing task of FeasPar, and are therefore supplemented in this work. PARSEC required a highly domain specific mapper to work within a translation system, whereas FeasPar outputs interlingua directly. Further, PARSEC only deliv-

[1] For technical reasons, these comparison were made in disfavor of FeasPar. FeasPar's performance would have been 10 % (estimated) better if measured under equal conditions, see Section 6.2.2

```
([statement]
 ([clause]
   ([misc] then)
   ([iaux] let+s)
   ([action] plan)
   ([patient] it)
   ([time] for then)
   ([mod-1] on the eighth)
   ([mod-1] eight)
   ([mod-1] to ten)))
```

Figure 7.1: PARSEC parse for "then let+s plan it for then on the eighth eight to ten".

```
((speech-act *suggest)
 (sentence-type *directive)
 (frame *schedule)
 (what ((frame *it)))
 (when ((frame *time-list)
        (connective -)
        (items (*MULTIPLE*
                ((frame *simple-time)
                 (day 8))
                ((frame *interval)
                 (incl-excl inclusive)
                 (start ((frame *simple-time)
                         (hour 8)))
                 (end ((frame *simple-time)
                       (hour 10)))))))))
 (conjunction then))
```

Figure 7.2: FeasPar parse for "then let +s plan it for then on the eighth eight to ten".

ers one parse hypothesis, whereas FeasPar can deliver an N-best-list of parse hypotheses ranked by probabilities.

SCREEN performs a syntactic and semantic analysis of spontaneous speech. Time Scheduling Task evaluation results are only given for some submodules, and not for the overall system. My best-case estimate yields an overall perfor-

mance of at most 35 % (see Section 2.3.5 on page 17) on the Time Scheduling Task. In comparison, FeasPar has a performance of 60.5 %. It is not defined as to what kind of output formalism SCREEN has, and not discussed or described how to apply it to other NLP system components, e.g. a generator. In contrast, FeasPar has well known and highly descriptive parse output formalism. It is clearly shown in this thesis that integration with a generation grammar for a target language is trivial.

7.2 Suitability for Various Tasks

This thesis gives clear evidence for the successful application of FeasPar as a semantic parser, extracting the semantic meaning (ESST). Additionally, a syntactic labeling task for German is well mastered (CR). Since German is known for its complex morpho-syntax, this shows that FeasPar is also suited for syntax parsing tasks. Further, it suggests that FeasPar's architecture is language independent.

Feature structures as representation formalism play a dominant role within classical computational linguistics, due to its powerful means of representing all types of linguistic information. This expressiveness is a major reason for the popularity of unification based grammars. FeasPar utilizes feature structures as parse output formalism, allowing for complex analysis descriptions with high information content. FeasPar makes no architectural constraints whether on the number of features nor on the depth in the structures. Both are learned from the domain training examples. Hence, this thesis suggests that FeasPar is applicable to various domains and various analysis needs.

FeasPar is easily integrated to other NLP components, due to its standard input (sentence hypothesis) and its standard output (feature structure). JANUS exemplifies this integration, where the speech recognizer feeds FeasPar, which again can feed a target language generator directly.

7.3 Ease of Use for Non-Experts

Since the trend in NLP has been going from general, domain-independent systems to specialized systems targeted towards a special domain, the importance of portability has grown. The central issue is the time required to develop an NLP system for a new domain. To parse (often) malformed input is a hard task. Grammar development is often the most time consuming task in building a total natural language understanding or translation system. When developing parse grammars, in theory, one needs a domain independent kernel, and a domain specific part, enabling reuse of the kernel in new domains. In practice, however, for various reasons, like the need to reduce parse ambiguities, the entire parse grammar is developed from scratch, requiring a lot of time from a grammar writer

expert. Further, experience from the JANUS project also shows that much time is spent by grammar writers and generator writers to adjust their common interface, the interlingua, so that grammar and generator development tasks are made as easy as possible. With FeasPar, the person in charge of the parser avoids these time consuming human processes: neither grammar theory understanding nor time for grammar development are required. Further, no agreements with generation grammar developers and interlingua adjustments are necessary. The only requirements are passive language competence, i.e. knowing the language just well enough to understand the semantics of various parts of a sentence, and an understanding of what various interlingua parts mean. For a person with this knowledge it will take two weeks to do the hand modeling effort, which includes modeling a lexicon and aligning training sentences with the ILTs, which are available from the generator grammar developers. They have to agree on an interlingua and a hand produced set of interlingua examples for the purpose of their own work. Also, their work is made easier as they do not have to consider the needs of the parse grammar developers in interlingua design. Further, the number of training sentences needed for FeasPar's learning process is small, only approximately 630 for ESST. This contributes to keeping the human effort low.

Automatic architecture set-up takes care of the division and distribution of the parsing task over several neural networks. FeasPar generates training data and all necessary architectural parameters, both for training and for run-time.

7.4 Other Advantages

The automatic architecture set-up typically splits the parse task into many networks (for ESST: 44 networks). Fortunately, these can be trained independently of each other without any need of communication between the training processes, allowing for parallel training on separate CPUs. In run-time, parallel parsing is in principle possible[2]. As soon as the three Chunker networks are done, the input for all remaining networks is fixed. Hence, the remaining networks could be run in parallel on separate CPU's, and then the results could be collected on one machine, which performs the search.

Another spin-off result from the generator grammar development is the interlingua specification. FeasPar uses it in its efficient Consistency Checking Search, so that it is guaranteed that the parse is consistent, and does not cause unexpected problems for the generators. Surprisingly, consistency checking is not available in any of the symbolic or connectionist parsers known to the author.

FeasPar delivers a probability score along with the parse, based on the neural network activations. The search guarantees that the valid parse with the highest probability is delivered as a result. FeasPar experiments show that it is possible to produce N-best lists of valid parses, ranked by probability, and that their evaluation could increase the parse performance (see Section 6.4).

[2]This is not implemented, but the implementation is substantially easy.

Chapter 8

Conclusion

This chapter will sum up the thesis by describing contributions, shortcomings and future work.

8.1 Contributions of the Thesis

This thesis claims that it is possible to build a neural net based parser that has a performance similar to a good hand modeled unification based parser. The presented parser delivers feature structure parses, needs a small corpus and a minimum of hand modeling, learns, and is robust towards spontaneous speech and speech recognizer effects. This section will in short discuss these characteristics.

1. **High Performance - Low Cost:** By evaluating FeasPar with JANUS, it was shown that FeasPar achieves similar performance as a good unification grammar (GLR*) and a good RTN grammar (Phoenix) without requiring tedious grammar development. FeasPar required only 2 weeks for training data and lexicon preparations, compared with 2 years for GLR* and 9 months for Phoenix.

2. **Learning to Parse Spontaneous Speech:** FeasPar is the first learning parser clearly stating generalization results on a spontaneous speech task. [1]

3. **Robustness to Spontaneous Speech and Speech Recognizer Errors:** FeasPar is the first parser being robust towards both spontaneous

[1] Jain presents results for PARSEC evaluated for read speech. These include generalization results for transcribed data and training results for speech data. Wermter and Weber present results for SCREEN evaluated with spontaneous speech data, but only at module level, not as overall performance.

speech effects and speech recognizer errors, without needing explicit architectural or hand modeled constructs[2] for this. Also, the training set contains no speech recognizer error examples. All these types of malformation are tolerantly handled by the FeasPar networks and the Consistency Checking Search.

4. **Dividing Feature Structure into Smaller, Learnable Problems:** The Chunk'n'Label principle introduced in this thesis demonstrates how a feature structure can be reformulated as a tree, how feature pairs and paths are defined, and aligned to the natural language chunks. Further, it is shown how the separate neural networks are defined with their output representations, their training data, and the lexicon. Finally, it is demonstrated how the interaction of the networks work in run time.

5. **Various Learning Techniques:** The learning techniques applied in PARSEC are insufficient for the more complex task of FeasPar, and are therefore supplemented with the LNC, SCS, and extended context connectivity principles as well as with statistical microfeatures in the lexicon. They all positively influence learning performance. Two other techniques (hybrid encoding and 2nd parse) have a negative influence on the ESST learning performance, due to the little and highly irregular training data. With larger training data, these might be valuable.

6. **Consistency Checking Search:** To my knowledge, no other parser performs a consistency check of its parsing result. Four advantages are offered by the Consistency Checking Search processing: first, FeasPar only outputs valid parses, and thereby guarantees other NLP components (the target language generators in JANUS) that their input fulfills the formalism definition; second, the parse performance increases considerably, since obviously wrong parses are sorted out; third, a parse probability is returned along with the parse; and fourth, an N-best list ranked with probabilities may be returned as parse result, allowing NLP components, e.g. discourse analyzers, to apply further constraints.

8.2 Shortcomings

The FeasPar's lexicon contains no lexical disambiguation, so that words like schedule are represented both as a verb and as a noun, and this ambiguous representation is presented to the networks. They must apply the correct interpretation and produce the right feature structure information, meaning that

[2]Symbolic parsers contain grammar modules and grammar-independent modules dedicated for robustness handling. The connectionist parser SCREEN contains modules specialized for various repair tasks.

the networks must be robust towards ambiguities too. The connectionist representation of lexical ambiguities is modeled by setting all features that are valid in one or more interpretations. Especially English contains many lexically ambiguous words, since verb and noun forms are often identical. This is clearly illustrated by parsing the sentence "time flies like an arrow". Even if the FeasPar evaluation performance is good for the English Spontaneous Speech Task, the performance might be even better if the networks do not have to be robust towards lexically ambiguous input.

8.3 Future Work

Certain aspects of FeasPar could be further investigated and expanded. Suggestions for future work will be discussed in this section.

- **Implementing Parallel Network Processing for Parse Mode:** Currently, an average sentence takes about 7 sec to parse, from which more than four fifth of the time is spent on running neural networks. [3] Since most of these neural networks can run independently of each other, one could let these be processed in parallel on several CPU's.
- **Implementing Utterances as Input instead of Sentences:** The current FeasPar implementation takes separate sentences as input. One utterance contains several sentences. To make it work with utterances, there are two options; first, integrate the GLR* sentence breaker with FeasPar, since the breaker is currently an external preprocessor when utterances are to be parsed by FeasPar; or second, introduce another chunk level and learn sentence breaks, which means extending FeasPar with one more chunker network and utterance feature networks.
- **Performance Improvement Through Rescoring :** Currently, FeasPar is able to produce N-best lists of valid parses ranked by probability. An initial experiment shows a potential for reducing the parse error rate to the half, if a rescoring would rank the N-best candidates differently. One important, so far unused knowledge source to draw on for an alternative criterion, could be the statistical consistency, discussed in Section 5.1.1.
- **More Training Data:** In general, more training data normally increases evaluation performance. In FeasPar, the increase may be especially large, since more training data would probably also enable a positive performance contribution from the cooperative networks, hybrid encoding and second parse.
- **Lattice or N-best Hypothesis Parsing:** FeasPar currently only utilizes the speech recognizer hypothesis with the highest score from acoustics

[3] When using one CPU on an Alpha Server 2100 4/275, 512 MB RAM

and language modeling. It would be interesting to parse several hypotheses, either as N-best lists or as speech lattices. The first is trivial to implement, the second is not. However, analyzing several hypotheses causes parse time to rise significantly.

- **Integration of Acoustic Scores:** Since the speech recognizer produces a score for a speech hypothesis, it would be valuable to utilize it in FeasPar, either as part of the input sentence, or as a mixture with the probability score of every feature structure.

Appendix A

ESST Features

The ESST feature that FeasPar learns:

no.	*Feature name*	*Feature values*
0	speech-act	*opening *address *suggest *accept *acknowledge *state-constraint *reject *confirm *request-response *closing *affirm *negate *request-suggestion *confirm-time
1	sentence-type	*fixed-expression *names *state *query-if *directive *query-ref *fragment
2	frame	*greet *hello *address *person-name *schedule *we *meeting *special-time *length *respond *booked *i *something *simple-time *interval *meet *you *clarify *class *considering *interject *adverb *busy *time-list *how *that *free *it *restaurant-name *thank *seminar *out-of-town *babble *settled *let-me-check *see-you *exclaim *return *what *relative-time *apologize *calendar *takecare *city-name *length-list *inform *know *conference-room *bye *event-list *needed *wait *appointment *here *look-forward *haveanice *undesired *check *lunch *pro *they *tennis *event-time
3	type	(COMPLEX VALUES)
4	first-name	(open class)
5	attitude	*should *possible *desired *needed *how-about *undesired *shall
6	who	(COMPLEX VALUES)
7	specifier	indefinite sometime definite next 2 plural at-least that anytime brunch all-range late following perhaps what approximate couple early this only another other even any the-rest-of most-member more-than all-member a-lot except 3 also first full both-of either-of third right most-range negative
8	what	(COMPLEX VALUES)
9	name	(open class)

no.	*Feature name*	*Feature values*
10	when	(COMPLEX VALUES)
11	unit	hour week
12	quantity	2 1 couple how-many 1.5 short-time how-long 2.5
13	how-long	(COMPLEX VALUES)
14	type	affirmative yes no negative i-dont-know
15	degree	weak normal superlative strong
16	adverb	babble perhaps actually really only if-possible again generally also already just right-now completely still unfortunately sort-of after-all surely always barely definitely
17	day-of-week	(open class)
18	topic	(COMPLEX VALUES)
19	incl-excl	inclusive exclusive
20	hour	(open class)
21	start	(COMPLEX VALUES)
22	end	(COMPLEX VALUES)
23	time-of-day	morning afternoon evening
24	whose	(COMPLEX VALUES)
25	day	(open class)
26	clarified	(COMPLEX VALUES)
27	conjunction	so therefore and then because if but or unless
28	connective	- and or eor between
29	items	(COMPLEX VALUES)
30	am-pm	pm am
31	minute	(open class)
32	month	(open class)
33	where	(COMPLEX VALUES)
34	with-whom	(COMPLEX VALUES)
35	babble	lets-see well
36	why	(COMPLEX VALUES)
37	exclaim	geez gee goodness my god
38	length	(COMPLEX VALUES)
39	direction	+
40	origin	(COMPLEX VALUES)
41	to-whom	(COMPLEX VALUES)
42	of	(COMPLEX VALUES)
43	title	(open class)
44	last-name	(open class)
45	period	day
46	from	(COMPLEX VALUES)
47	event	(COMPLEX VALUES)

Bibliography

[All88a] J. Allen. Basic Parsing Techniques. In *[All88b]*, chapter 2. The Benjamin/Cummings Publishing Company, Inc., 1988.

[All88b] J. Allen. *Natural Language Understanding.* The Benjamin/Cummings Publishing Company, Inc., 1988.

[BBD+95] Samuel Bayer, Erica Bernstein, David Duff, Lynette Hirschman, Susann LuperFoy, and Margot Peet. Spoken Language Understanding: Report on the MITRE Spoken Language System. In *Proceedings of ARPA Spoken Language Systems Technology Workshop*, 1995.

[BBMG+94] S. K. Bennacef, H. Bonneau-Maynard, J. L. Gauvain, L. Lamel, and W. Minker. A Spoken Language System For Information Retrieval. In *Proceedings of the International Conference on Spoken Language Processing - ICSLP*, Yokohama, sept 1994.

[BCP+90] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A Statistical Approach To Machine Translation. *Computational Linguistics*, 16(2):79–85, June 1990.

[Ber91] George Berg. Learning Recursive Phrase Structure: Combining the Strengths of PDP and X-Bar Syntax. Technical report TR 91-5, Dept. of Computer Science, University at Albany, State University of New York, 1991.

[BIS92] Robert Bobrow, Robert Inria, and David Stallard. Syntactic/Semantic Coupling in the BBN DELPHIi System. In *Proceedings of ARPA Spoken Language Systems Technology Workshop*, 1992.

[BLM91a] Peter F. Brown, Jennifer C. Lai, and Robert L. Mercer. Aligning Sentences In Parallel Corpora. In *Proceedings 29th Annual Meeting of the Association for Computational Linguistics*, pages 169–176, Berkeley,CA, 1991.

[BLM91b] Peter F. Brown, Jennifer C. Lai, and Robert L. Mercer. Word-sense Disambiguation Using Statistical Methods. In *Proceedings 29th Annual Meeting of the Association for Computational Linguistics*, pages 264–270, Berkeley,CA, 1991.

[BPP+91] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Robert L. Mercer, and Surya Mohanty. Dividing and Conquering Long sentences in a Translation System. Technical report, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1991.

[BPW94] Finn Dag Buø, Thomas Polzin, and Alex Waibel. Learning Complex Output Representations In Connectionist Parsing of Spoken Languages. In *International Conference on Acoustics, Speech & Signal Processing*, pages 365–368, vol. 1, Adelaide, Australia, April 1994. IEEE.

[Bre82] J. Bresnan, editor. *The Mental Representation of Grammatical Relations*. The MIT Press, Cambridge, MA, 1982.

[Buø92] Finn Dag Buø. A Learnable Connectionist Parser that Outputs Feature Structures. Ph.D proposal, Fakultät für Informatik, Universität Karlsruhe, Germany, November 1992.

[BW96a] Finn Dag Buø and Alex Waibel. FeasPar - A Feature Structure Parser Learning to Parse Spoken Language. In *Proceedings of the International Conference on Computational Linguistics*, August 1996.

[BW96b] Finn Dag Buø and Alex Waibel. Learning To Parse Spontaneous Speech. In *Proceedings of the International Conference on Spoken Language Processing*, October 1996.

[BW96c] Finn Dag Buø and Alex Waibel. Search in a Learnable Spoken Language Parser. In *Proceedings of the 12th European Conference on Artificial Intelligence*, August 1996.

[CH83] Jaime G. Carbonell and Philip J. Hayes. Recovery Strategies for Parsing Extragrammatical Language. *American Journal of Computational Linguistics*, 9(3-4):123–146, 1983.

[Chr91] Lonnie Chrisman. Learning Recursive Distributed Representations for Holistic Computation. *Connection Science*, 3(4):345–366, 1991.

[CMGS91] Anna Corazza, Renato De Mori, Roberto Gretter, and Giorgio Satta. Computation of Probabilities for an Island-Driven Parser. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):936–950, 1991.

[DCG93] Ido Dagan, Kenneth W. Church, and William A. Gale. Robust Bilingual Word Alignment for Machine Aided Translation. In *Proceedings of the Workshop on Very Large Corpora at ACL*, 1993.

[Elm90] J. L. Elman. Finding Structure in Time. *Cognitive Science*, 14:179–211, 1990.

[Elm91] Jeffrey L. Elman. Distributed Representations, Simple Recurrent Networks, and Grammatical Structure. *Machine Learning*, pages 195–225, 1991.

[ER68] E.Bach and R.Harms, editors. *Universals in Linguistic Theory*. Holt, Rinehart and Winston, New York, 1968.

[FI92] Osamu Furuse and Hitoshi Iida. Cooperation between Transfer and Analysis in Example-Based Framework. In *Proceedings of COLING*, 1992.

[Fil68] Charles J. Fillmore. The case for case. In *[ER68]*, pages 1–88. Holt, Rinehart and Winston, New York, 1968.

[FSI92] Osamu Furuse, Eiichiro Sumita, and Hitoshi Iida. Building Transfer Knowledge from a Bilingual Spoken-Dialogue Corpus. *ACL*, 13(5), 1992.

[GC93] William A. Gale and Kenneth W. Church. A Program for Aligning Sentences in Bilingual Corpora. *Computational linguistics*, 19(1):75–91, 1993.

[GKPS85a] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. A theory of syntactic features. In *[GKPS85b]*, chapter 2. Blackwell Publishing, Oxford, England and Harvard University Press, Cambridge, MA, USA, 1985.

[GKPS85b] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammar*. Blackwell Publishing, Oxford, England and Harvard University Press, Cambridge, MA, USA, 1985.

[GSB+95] P. Geutner, B. Suhm, F. D. Buø, T. Kemp, L. Mayfield, A. E. McNair, I. Rogina, T. Schultz, T. Sloboda, W. Ward, M. Woszczyna, and A. Waibel. Integrating Different Learning Approaches into a Multilingual Spoken Language Translation System. In *Workshop on New Approaches to Learning for Natural Language Processing, International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

[Hen94] James B. Henderson. *Description Based Parsing in a Connectionist Network*. PhD thesis, University of Pennsylvania, 1994.

[HKP91a] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[HKP91b] John Hertz, Anders Krogh, and Richard G. Palmer. Multi-Layer Networks. In *[HKP91a]*, chapter 6. Addison-Wesley, 1991.

[HKP91c] John Hertz, Anders Krogh, and Richard G. Palmer. The Hopfield Model. In *[HKP91a]*, chapter 2. Addison-Wesley, 1991.

[HMR86] Geoffrey E. Hinton, J. L. McClelland, and David E. Rumelhart. Distributed Representations. In *[RMtPrg86]*. The MIT Press, 1986.

[HS86] G. E. Hinton and T. J. Sejnowski. Learning and Relearning in Boltzmann Machines. In *[RMtPrg86]*. The MIT Press, 1986.

[IW93] Sunil Issar and Wayne Ward. CMU's robust spoken language understanding system. In *Proceedings of Eurospeech*, 1993.

[Jai89] Ajay Jain. A Connectionst Architecture for Sequential Symbolic Domains. Technical report, School of Computer Science, Carnegie Mellon University, Pitt. PA, USA, 1989.

[Jai90] Ajay Jain. Parsing Complex Sentences with Stuctured Connectionist Networks. In Jeffrey Elman, editor, *Neural Computation 3*, pages 110–120. MIT, School of Computer Science, Carnegie Mellon University, Pitt. PA, USA, 1990.

[Jai91] Ajay N. Jain. *A Connectionist Learning Architecture for Parsing Spoken Language*. PhD thesis, School of Computer Science, Carnegie Mellon University, Dec 1991.

[Jai92] Ajay N. Jain. Generalization Performance in PARSEC - A Structured Connectionist Parsing Architecture. In J.E.Moddy, S.J.Hanson, and R.P.Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann Pub., 1992.

[Jel90] F. Jelinek. Self-Organized Language Modeling For Speech Recognition. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*. Morgan Kaufmann, San Mateo, CA, USA, 1990.

[JJM92] F. Jelinek, J.D.Lafferty, and R.L. Mercer. Basic Methods of Probabilistic Context Free Grammars. In P.Laface and R. De Mori, editors, *Speech Recognition and Understanding. Recent Advances*, pages 345–360, Berlin Heidelberg, 1992. NATO ASI Series, Vol F.75, Springer-Verlag.

[JLM+94] F. Jelinek, J. Lafferty, D. Magerman, L. R. Mercer, A. Ratnaparkhi, and S. Roukos. Decision Tree Parsing Using a Hidden Derivation Model. In *Proceedings ARPA Workshop on Human Language Technology*, pages 260–265, Princeton, New Jersey, March 1994.

[JW89] Ajay Jain and Alex Waibel. A Connectionist Parser Aimed at Spoken Language. Technical report, School of Computer Science, Carnegie Mellon University, Pitt. PA, USA, august 1989.

[JW90a] Ajay Jain and Alex Waibel. Incremental Parsing by Modular Recurrent Connectionist Networks. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA, USA, Computer Science, Carnegie Mellon University, Pitt. PA, USA, 1990.

[JW90b] Ajay Jain and Alex Waibel. Robust Connectionst Parser of Spoken Language. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. School of Computer Science, Carnegie Mellon University, Pitt. PA, USA, 1990.

[KB82] R. Kaplan and J. Bresnan. Lexical-Functional Grammar: A Formal Sysytem for Grammatical Representation. In *[Bre82]*, pages 173–281. The MIT Press, Cambridge, MA, 1982.

[KK93] Christel Kemke and Habibatou Kone. INCOPA - An Incremental Connectionist Parser. In *Proceedings of Would Congress on Neural Networks*, pages 41–44, Portland, Oregon, 1993.

[KR93] Martin Kay and Martin Röscheisen. Text-Translation Alignment. *Computational linguistics*, 19(1):120–142, 1993.

[KS93] Christel Kemke and Christoph Schommer. PAPADUES - Parallel Parsing of Ambiguous Sentences. In *Proceedings of Would Congress on Neural Networks*, pages 79–82, Portland, Oregon, 1993.

[Lan90] Hagen Langer. Syntaktische Normalisierung gesprochener Sprache. Arbeitsbericht nr.23, DFG-Forschergruppe Kohärenz, Fakultät für Linguistik und Literaturwissenschaft der Universität Bielefeld, 1990.

[Lav96a] A. Lavie. personal communication. e-mail, Jan 1996.

[Lav96b] Alon Lavie. *GLR*: A Robust Focused Parser for Spontaneously Spoken Language*. PhD thesis, CMU, to appear 1996.

[LFG95a] Steve Lawrence, Sandiway Fong, and C. Lee Giles. On the Applicability of Neural Network and Machine Learning Methodologies to Natural Language Processing. In *Workshop on New Approaches to Learning for Natural Language Processing, International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, August 1995.

[LFG+95b] Bruce Lund, William M. Fischer, John S. Garofolo, David S. Pallett, Mark Przybocki, and R. Allen Wilkinson. A Spoken Natural Language Interface To Libraries. In *Proceedings of ARPA Spoken Language Systems Technology Workshop*, 1995.

[LGQ+95] Lori Levin, Oren Glickman, Yan Qu, Donna Gates, Alon Lavie, Carolyn P. Rose, Carol Van Ess-Dykema, and Alex Waibel. Using Context in Machine Translation of Spoken Language. In *Proceedings of Theoretical and Methodological Issues in Machine Translation*, 1995.

[LR90] Sebastian Lisken and Hannes Rieser. Ein inkrementeller Parser zur Analyse von simulierten Reparaturen ("repairs"). Arbeitsbericht nr. 29, DFG-Forschergruppe Kohärenz, Fakultät für Linguistik und Literaturwissenschaft der Universität Bielefeld, 1990.

[LT93] A. Lavie and M. Tomita. GLR* - An Efficient Noise-skipping Parsing Algorithm for Context-free Grammars. In *Proceedings of Third International Workshop on Parsing Technologies*, pages 123–134, 1993.

[MBB+95] S. Miller, M. Bates, R. Bobrow, R. Ingria, J. Makhoul, and R. Schwarz. Recent Progress in Hidden Understanding Models (HUM). In *Proceedings of the ARPA Spoken Language Systems Technology Workshop*, 1995.

[MBSI94] S. Miller, R. Bobrow, R. Schwarz, and R. Ingria. Statistical Language Processing Using Hidden Understanding Models. In *Proceedings of the ARPA Spoken Language Systems Technology Workshop*, 1994.

[MD89a] R. Miikkulainen and M. Dyer. A Modular Neural Network Architecture for Sequential Paraphrasing of Script-Based Stories. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 1989.

[MD89b] R. Miikkulainen and M. Dyer. Encoding Input/Output Representations in Connectionist Cognitive Systems. In D.S.Touretzky, G.E.Hinton, and T.J.Sejnowski, editors, *Proceedings of the 1988*

Connectionist Models Summer School. Morgan Kaufmann Publishers, Los Altos, CA, 1989.

[MD91] R. Miikkulainen and M. Dyer. Natural Language Processing With Modular PDP Networks and Distributed Lexicon. *Cognitive Science*, 15:343–399, 1991.

[MG93] M. McCandless and J. R. Glass. Decision Tree Parsing Using a Hidden Derivation Model. In *Proceedings ARPA Workshop on Human Language Technology*, pages 981–984, 1993.

[MGS+95] L. Mayfield, M. Gavalda, Y-H. Seo, B. Suhm, W. Ward, and A. Waibel. Parsing Real Input in JANUS: A Concept-Based Approach to Spoken Language Translation. In *Proceedings of TMI, Leuven*, 1995.

[MGWW95] L. Mayfield, M. Gavalda, W. Ward, and A. Waibel. Concept-Based Speech Translation. In *ICASSP 95*, pages 97–100. IEEE, 1995.

[Min95] Wolfgang Minker. An English Version of the LIMSi L'ATIS System. Technical Report Notes et documents LIMSI No. 95-12, Laboratoire d'Informatique pour la Mecanique et les Sciences de l'Ingenieur, LIMSI - CNRS, apr 1995.

[MNC91] Teruko Mitamura, Eric H. Nyberg, and Jaime G. Carbonell. An Efficient Interlingua Translation System for Multilingual Document Production. In *Proceedings of the Machine Translation Summit III*, Washington DC, july 1991.

[MST+92] Tsuyoshi Morimoto, Masami Suzuki, Toshiyuki Takezawa, Gen'ichiro Kikui, Masaaki Nagata, and Mutsuko Tomokiyo. A Spoken Language Translation System: SL-TRANS2. In *Proceedings of COLING-92*, pages 1048–1052. ATR Intepreting Telephony Research Laboratories, Aug. 23-28. 1992.

[Net92] Klaus Netter. On Non-Head Non-Movement. In *Proceedings of Konvens 92*, pages 218–227. Springer-Verlag, Berlin Heidelberg, 1992.

[NNP92] John Nerbonne, Klaus Netter, and Carl Pollard. German Grammar in HPSG. In *CSLI Lecture Notes*. Chicago, 1992.

[NS92] Sven Naumann and Jürgen Schrepp. An empirical approach to syntax learning. In Günther Götz, editor, *Proceedings of KONVENS92*, pages 209–217. Gesellschaft für Informatik, Springer-Verlag, Oct 1992.

[NT87] See-Kiong Ng and Masaru Tomita. Probabilistic LR Parsing for General Context-free Grammars. Technical report, Center for Machine Translation, CMU, 5000 Forbes Ave., Pittsburgh, PA 15213 USA, 1987.

[OAM+92] Louise Osterholtz, Charles Augustine, Artur McNair, Ivica Rogina, Hiroaki Saito, Tilo Sloboda, Joe Tebelskis, and Alex Waibel. Testing Generality In JANUS: A Multi-Lingual Speech Translation System. In *Proceedings of ICASSP*. IEEE, 1992.

[Par92] M. Paritong. Constituent Coordination in HPSG. In *Proceedings of Konvens 92*, pages 228–237. Springer-Verlag, Berlin Heidelberg, 1992.

[PBA93] Lutz Prechelt, Finn Dag Buø, and Rolf Adams. Transportable Natural Language Interfaces for Taxonomic Knowledge Representation Systems. In *Conference on Artificial Intelligence Applications*, Orlando, Florida, March 1993. IEEE.

[Pol88] J. B. Pollack. Recursive Auto-Associative Memory: Devising Compositional Distributed Representations. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.*, Hillsdale, NJ, 1988. Lawrence Erlbaum.

[PS87a] C. Pollard and I. Sag. *An Information-Based Syntax and Semantics*. CSLI Lecture Notes No.13, 1987.

[PS87b] C. Pollard and I. Sag. Formal Foundations. In *[PS87a]*, chapter 2. CSLI Lecture Notes No.13, 1987.

[PW92] T. S. Polzin and A. Waibel. Learning the ATIS-Task. Technical report, Carnegie Mellon University, 1992.

[Rab90] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, 1989. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*. Morgan Kaufmann, San Mateo, CA, USA, 1990.

[Rec93] Christine Reck. Robustes Parsen von Dialogen mit semantischen Grammatiken. Studieanarbeit, Fakultät für Informatik, Universität Karlsruhe, Germany, Jan 1993.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. In *[RMtPrg86]*. The MIT Press, 1986.

[RMtPrg86] David Rumelhart, James L. McClland, and the PDP research group, editors. *Parallel Distributed Processing.* The MIT Press, 1986.

[SB92] David Stallard and Robert Bobrow. Fragment Processing in the DELPHI System. In *Proceedings of ARPA Spoken Language Systems Technology Workshop*, 1992.

[Sch92] Hinrich Schütze. Word Space. In J.E.Moody, S.J.Hanson, and R.P.Lippman, editors, *Advances in Neural Information Processing Systems 4.* Morgan Kaufmann Publishers, 1992.

[Sch93] Hinrich Schütze. Translation by Confusion. In *Spring Symposium on Machine Translation.* AAAI, 1993.

[Sen92] Stephanie Seneff. TINA: A Natural Language System for Spoken Language Applications. *Computational linguistics*, 18(1), 1992.

[Sha91] Noel E Sharkey. Connectionist Representation Techniques. Technical Report R 217, Centre for Connection Science, Department of Computer Science, University of Exeter, 1991.

[SL92] K. Sikkel and M. Lankhorst. A Parallel Bottom-Up Tomita Parser. In *Proceedings of Konvens 92*, pages 238–247. Springer-Verlag, Berlin Heidelberg, 1992.

[Sto93] Andreas Stolcke. An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities. Technical Report TR-93-065, ICSI, Berkeley, CA, 1993.

[TC87] M. Tomita and J. Carbonell. The Universal Parser Architecture for Knowledge-Based Machine Translation. Technical report CMU-CMT-87-101, Center for Machine Translation, CMU, 5000 Forbes Ave., Pittsburgh, PA 15213 USA, 1987.

[TMML88] M. Tomita(ed.), Teruko Mitamura, Hiroyuki Musha, and Marion Lee. The Generalized LR Parser/Compiler Version 8.1: User's Guide. Technical report CMU-CMT-88-MEMO, Center for Machine Translation, CMU, 1988.

[Tom85] M. Tomita. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Boston, MA, 1985.

[Tom87] Masaru Tomita. An Efficient Augmented-Context-Free Parsing Algorithm. *Computational Linguistics*, 13(1-2):31–46, 1987.

[Tr88] M. Tomita and Eric H. Nyberg 3rd. Generation Kit and Transformation Kit Version 3.2: User's Manual. Technical report CMU-CMT-88-MEMO, Center for Machine Translation, CMU, 1988.

[TSP+95] Dinesh Tummala, Stephanie Seneff, Douglas Paul, Clifford Weinstein, and Dennis Yang. CCLINC: System Architechure and Concept Demonstration of Speech-to-Speech Translation for Limited-Domain Multilingual Applications. In *Proceedings of ARPA Spoken Language Systems Technology Workshop*, 1995.

[Usz86a] Hans Uszkoreit. Constraints on Order. Technical Report CSLI-86-46, Center for the Study of Language and Information, Leland Stanford Junior University, january 1986.

[Usz86b] Hans Uszkoreit. Linear Precedence in Discontinuous Constituents: Complex Fronting in German. Technical Report CSLI-86-47, Center for the Study of Language and Information, Leland Stanford Junior University, january 1986.

[Usz87] Hans Uszkoreit. *Word order and constituent structure in German.* Center for the study of language and information, CSLI/Stanford, Ventura Hall, Stanford, CA 94305, 1987.

[War91] Wayne Ward. Understanding Spontaneous Speech: The Phoenix System. In *ICASSP 91*, pages 365–367. IEEE, 1991.

[WAWB+94] M. Woszczyna, N. Aoki-Waibel, F. D. Buø, N. Coccaro, K. Horiguchi, T. Kemp, A. Lavie, A. McNair, T. Polzin, I. Rogina, C.P. Rose, T. Schultz, B. Suhm, M. Tomita, and A. Waibel. JANUS 93: Towards Spontaneous Speech Translation. In *International Conference on Acoustics, Speech & Signal Processing*, pages 345–348, vol. 1, Adelaide, Australia, April 1994. IEEE.

[WBB+94] A. Waibel, U. Bodenhausen, F. D. Buø, N. Coccaro, H. Hild, T.S. Polzin, and B. Suhm. Connectionist Modules in a Multi-Lingual Speech Translation System. In *International Conference on Neural Information Processing (ICONIP)*, Seoul, Korea, October 1994. IEEE.

[Wen93] Fuliang Weng. Handling Syntactic Extra-Grammaticality. In *Proceedings of Third International Workshop on Parsing Technologies*, pages 319–331, 1993.

[WFT89] Lars Holter Walter F. Tichy, Rolf Adams. NLH/E: A Natural Language Help System. In *Proceedings of the 11th International Conference on Software Engineering*, 1989.

[WI95] Wayne Ward and Sunil Issar. The CMU ATIS System. In *Proceedings of ARPA Spoken Language Systems Technology Workshop*, 1995.

[WJM+91] Alex Waibel, Ajay Jain, Arthur McNair, Joe Tebelskis, Hiroaki Saito, and Alexander G. Hauptmann. JANUS: A Speech-to-Speech Translation System Using Connectionist and Symbolic Processing Strategies. In *ICASSP*. IEEE, 1991.

[WJM+92] Alex Waibel, Ajay Jain, Arthur McNair, Joe Tebelskis, Louise Osterholtz, Hiroaki Saito, Otto Schmidbauer, Tilo Sloboda, and Monika Woszczyna. JANUS: Speech-to-Speech Translation Using Connectionist and Non-Connectionist Techniques. In J.E.Moody, S.J.Hanson, and R.P.Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann Publishers, 1992.

[WMG+96] Monika Woszczyna, Laura Mayfield, Marsal Gavalda, Matthias Denecke, Christine Reck, and Andreas Eisele. personal communication, 1993-96.

[WNM+91] Alex Waibel, Ajay N.Jain, Arthur E. McNair, Hiroaki Saito, Alexander G. Hauptmann, and Joe Tebelskis. JANUS: A Speech-To-Speech Translation System Using Connectionist And Symbolic Processing Strategies. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 1991.

[WP85] D. Waltz and J. Pollack. Massively parallell parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9, 1985.

[WS90a] Rolf Wilkens and Helmut Schnelle. A connectionist parser for context-free phrase structure grammars. In G. Dorffner, editor, *Konnektionismus in Artificial Intelligence und Kognitionsforschung*. Springer, Sept 1990.

[WS90b] Rolf Wilkens and Helmut Schnelle. Konnektionistische Repräsentation von grammatischem Wissen. 1990.

[WS91] Rolf Wilkens and Helmut Schnelle. Representation of Principles and Parameters in a Connectionist Network. 1991.

[WW94] Stefan Wermter and Volker Weber. Learning Fault-tolerant Spreech Parsing with SCREEN. In *Proceedings of Twelfth National Conference on Artificial Intelligence*, Seattle, 1994.

[WW96] Stefan Wermter and Volker Weber. SCREEN: Learning a Flat Syntactic and Semantic Spoken Language Analysis using Artificial Neural Networks. *Journal of Artificial Intelligence Research*, submitted, 1996.

Lebenslauf

Geboren:	22. Juni 1966 in Stockholm, Schweden
Staatsangehörigkeit:	Norwegisch

Schulausbildung:

08/1973	-	06/1982	Grundschule, Asker, Norwegen
08/1982	-	06/1985	Gymnasium, Bærum, Norwegen
07/1985	-	07/1986	Wehrdienst, Norwegen

Universitätsausbildung:

09/1986	-	03/1991	Studium der Informatik an der Norwegischen Technischen Hochschule (NTH), Trondheim, Norwegen; Abschluß: Sivilingeniør (Diplom-Informatiker)

Studienaufenthalte im Ausland:

08/1989	-	06/1990	Technische Hochschule Linköping, Schweden
10/1990	-	03/1991	Universität Karlsruhe (TH), Institut für Programmstrukturen und Datenorganisation, Lehrstuhl Prof. Dr. Tichy Diplomarbeit

Praktische Tätigkeiten:

10/1984	-	06/1989	Customizing und Support, Kellydata, Norwegen
09/1987	-	05/1989	Tutor, NTH, Norwegen
06/1989	-	08/1989	Hotline-Support, Hewlett-Packard, Norwegen
06/1990	-	08/1990	Entwickler, Hewlett-Packard, Palo Alto, CA, USA
09/1991	-	12/1992	Stipendiat/Wissenschaftlicher Mitarbeiter, Institut für Programmstrukturen und Datenorganisation, Lehrstuhl Prof Dr. Tichy, Universität Karlsruhe (TH)
01/1993	-	06/1996	Stipendiat/Wissenschaftlicher Mitarbeiter, Institut für Logik, Komplexität und Deduktionssysteme, Lehrstuhl Prof. Dr. Waibel, Universität Karlsruhe (TH)
seit 03/1996			Entwicklungskoordinator HR-Norwegen, SAP AG, Walldorf