Verbundprojekt ASL
- Südverbund -

Architekturen von Systemen
zur integrierten Analyse von
Sprachlauten und Sprachstrukturen

Lehrstuhl für Informatik 5
(Mustererkennung)
Universität Erlangen-Nürnberg

Prof. Dr.-Ing. H. Niemann
Dr.-Ing. E. Nöth

Martensstr. 3
D–8520 Erlangen
(09131) 85–7774

Automatische Extraktion eines Sprachmodells für Zeitangaben aus einem ERNEST–Netzwerk und dessen Einsatz in der akustisch–phonetischen Dekodierung

S. Allwang
R. Kompe
E. Nöth
A. Kießling
T. Kuhn
M. Mast

ASL-Süd—TR—12—92/FAU

Juli 1992

Gehört zum Antragsabschnitt: 4.3 Interaktion Akustik–Linguistik

Die diesem Bericht zugrundeliegenden Vorhaben wurden mit Mitteln des Bundesministers für Forschung und Technologie unter dem Förderkennzeichen 01IV102H0 gefördert. Die Verantwortung für den Inhalt dieses Berichts liegt bei den Autoren.

Kurzfassung

Aus der linguistischen Wissensbasis des Sprachverarbeitungsystems EVAR wird eine kontextfreie Grammatik extrahiert, die die syntaktischen, semantischen, pragmatischen und dialogspezifischen Einschränkungen der Wissensbasis reporäsentiert. Die Wissensbasis ist in Form eines semantischen Netzes im ERNEST–Netzwerkformalismus abgelegt. Diese Grammatik wird bei der Worterkennung eingesetzt, um Konstituentenhypothesen zu generieren. In Kapitel 2 wird die Extraktion der kontextfreien Grammatik aus dem ERNEST–Netz erläutert. Nach einer kurzen Beschreibung des ERNEST–Netzwerkformalismus im Abschnitt 2.1 wird im Abschnitt 2.2 der Teil der Wissensbasis vorgestellt, der die Zeitangaben in EVAR modelliert. Im Abschnitt 2.3 schließlich wird der genaue Ablauf der automatischen Grammatikerzeugung dargelegt. Im Kapitel 3 folgt eine Beschreibung der Erzeugung von Konstituentenhypothesen unter Einsatz der extrahierten Grammatik. Aus der Grammatik wird durch den Compiler–Generator BISON ein Parser erzeugt, mit dessen Hilfe Wortketten syntaktisch korrekt erweitert werden können. Im Abschnitt 3.1 wird die Vorgehensweise des von BISON erzeugten $LALR(1)$–Parsers erläutert und wie dieser modifiziert werden muß, um bei der Generierung von Konstituentenhypothesen sinnvoll eingesetzt werden zu können. Im Abschnitt 3.2 folgt eine Beschreibung der automatischen Generierung von Konstituentenhypothesen. Kapitel 4 faßt die praktischen Ergebnisse der vorgestellten Algorithmen zusammen. Die Arbeit schließt mit einer Zusammenfassung.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eine der wichtigsten Formen der zwischenmenschlichen Verständigung ist gesprochene Sprache. Sie ist dem Menschen viel geläufiger als Sprache in handschriftlicher oder getippter Form. Heutzutage wird mit Computern kommuniziert, indem über eine Tastatur Befehle eingegeben werden, die der Rechner dann verarbeitet. Es vereinfacht den allgemeinen Umgang mit Rechnern jedoch wesentlich, wenn man sich mit ihnen mit Hilfe von normal gesprochener Sprache verständigen kann. Damit würde der Zugang zu digitalen Rechenmaschinen erleichtert werden.

Am Lehrstuhl für Informatik 5 (Mustererkennung) der Universität Erlangen–Nürnberg wird das sprachverstehende Dialogsystem EVAR (**E**rkennen, **V**erstehen, **A**ntworten, **R**ückfragen) entwickelt. Als Anwendungsgebiet wurde ein natürlichsprachlicher Auskunftsdialog über die Intercity–Züge der Deutschen Bundesbahn gewählt. Dabei wird kontinuierlich gesprochene Sprache in Telefonqualität sprecherunabhängig verarbeitet.

Die Komlpexität dieser Beispielanwendung wird anhand folgenden Protokolls eines real geführten Dialogs zwischen dem Benutzer O. und der Reiseauskunft B. verdeutlicht. Die Aufnahme entstand im Rahmen der Erstellung einer Dialogstichpobe ([Nöth89]).

B: Reiseauskunft Bahnfof Erlangen, Grüß Gott
O: Ja, hier ist Obermayer, Grüß Gott, ich hätt' eine Frage – ich möcht' morgen von Nürnberg nach Ulm fahren und möcht' ungefähr um 4 Uhr in Ulm sein. Wann muß ich da in Nürnberg wegfahren?
B: ... und zwar fahr'n Sie um 13 25 in Erlangen weg
O: 13 25 in Erlangen weg, ja ...
B: ... fahr'n bis nach Donauwörth ...
O: bis Donauwörth ...
B: sind dort um 15 Uhr 4 ...
O: 15 Uhr 4 in Donauwörth ...
B: ... und ab 29sten, also seit einer Woche gibt's den Anschlußzug wieder, um 15 9 geht's weiter ...
O: 15 Null 9 in Donauwörth weiter ...
B: ... der is' in Ulm 15 Uhr 53 ...
O: 15 Uhr 53 in Ulm – des paßt wunderbar, ... und in Nürnberg, wann fährt der in Nürnberg los?
B: In Nürnberg würde er 14 Uhr 8 weggehen ...
O: Um 14 Uhr 8 in Nürnberg los ...
B: Ja
O: Gut, bedanke mich, auf Wiederhören
B: Bitte, Wiederhören.

EVAR ist ein wissensbasiertes Analysesystem. Ein solches System enthält als wesentliche Komponente eine Wissensbasis, in der in irgendeiner Form das Wissen über den betrachteten Problemkreis abgelegt ist. In EVAR handelt es sich um linguistisches Wissen, wie zum Beispiel den syntaktischen Aufbau von Uhrzeitangaben. 'um zehn Uhr', 'um halb drei' sind syntaktisch korrekte Wortfolgen, 'neun um Uhr' dagegen nicht. Das für die Analyse des Sprachsignals notwendige linguistische Wissen ist in einem semantischen Netz [Quill68] abgelegt. Dieses Netz ist im ERNEST–Formalismus repräsentiert, einem Netzwerksystem, das ebenfalls am Lehrstuhl für Informatik 5 entwickelt wird (siehe [Sage90]). ERNEST ist dabei eine Abkürzung für **Er**langer **Ne**tzwerk**s**yst**e**m.

Das Sprachsignal wird wie folgt analysiert: Zunächst werden in der Erkennungsphase durch die

Worterkennung [Kunz89] Wörter im Sprachsignal positioniert und Worthypothesen erzeugt. Eine Worthypothese enthält das Wort, den zugehörigen Zeitbereich und eine Bewertung. Es kommen dabei stochastische Verfahren zum Einsatz. Um die potentiell gesprochenen Wörter mit großer Wahrscheinlichkeit in einer Äußerung zu finden, wird eine große Menge von konkurrierenden, sich zeitlich überlappenden Worthypothesen erzeugt. Daraus ergibt sich ein Hypothesengitter, in dem die möglichen Wortketten enthalten sind, die nicht notwendigerweise syntaktisch korrekt sein müssen. Die Verkettung von Worthypothesen zu Konstituenten- beziehungsweise Satzhypothesen erfolgt anschließend durch Graphsuchverfahren, wobei der Suchraum durch die Menge der Worthypothesen und eine Grammatik gegeben ist, die notwendiges Wissen repräsentiert [Kuhn89]. Mit Hilfe der Wissensbasis werden anschließend in der Verstehensphase die erzeugten Wortketten interpretiert. In der Wissensbasis selbst ist die Struktur der deutschen Sprache explizit beschrieben, wobei zwischen syntaktischen, semantischen, pragmatischen und dialogspezifischen Eigenschaften unterschieden wird. Es ist wünschenswert, möglichst frühzeitig das Wissen über die Sprachstruktur bei der Analyse des Sprachsignals einzusetzen. Zu diesem Zweck benötigt man ein Sprachmodell, das die zulässigen Restriktionen über der Menge der möglichen Äußerungen, die explizit in der Wissensbasis angegeben sind, in einem Grammatikformalismus vereinigt, der von der Worterkennung effizient genutzt werden kann.

Ziel der diesem Bericht zugrundeliegenden Arbeiten ist es, ein solches Sprachmodell aus dem ERNEST–Netzwerk, also aus der Wissensbasis, automatisch zu extrahieren. Dieses Sprachmodell wird als kontextfreie Grammatik dargestellt. Die Grammatik beschreibt dann die syntaktische Struktur von Wortfolgen, die gemäß den syntaktischen, semantischen, pragmatischen und dialogspezifischen Beschränkungen der Wissensbasis korrekte Äußerungen der deutschen Sprache darstellen. Aus Aufwandsgründen wurde die konkrete Modellbildung bisher auf die in EVAR modellierten Zeitangaben beschränkt. Aus der Grammatik und mit Hilfe eines Parsers wird dann das Sprachsignal wie folgt untersucht: Es werden alle Wörter bestimmt, mit denen gültige Zeitangaben beginnen dürfen. Diese Wörter werden schrittweise zu korrekten Ketten erweitert, indem zu jeder bereits ermittelten Teilkette eine Menge von Wörtern berechnet wird, mit denen die Teilkette syntaktische korrekt ergänzt werden kann. Diese Teilketten werden mit dem Sprachsignal verglichen und die Ähnlichkeit zwischen Wortkette und Sprachsignal bewertet. Aus den bestbewerteten syntaktisch vollständigen Ketten werden Wortkettenhypothesen, sogenannte Konstituentenhypothesen, generiert. Eine Konstituentenhypothese besteht aus einer identifizierenden Kettennummer, dem Anfang und dem Ende des zugehörigen Zeitbereichs, einer Bewertung und der entsprechenden Folge von Wörtern.

In Kapitel 2 wird die Extraktion der kontextfreien Grammatik aus dem ERNEST–Netz erläutert. Nach einer kurzen Beschreibung des ERNEST–Netzwerkformalismus im Abschnitt 2.1 wird im Abschnitt 2.2 der Teil der Wissensbasis vorgestellt, der die Zeitangaben in EVAR modelliert. Im Abschnitt 2.3 schließlich wird der genaue Ablauf der automatischen Grammatikerzeugung dargelegt. Im Kapitel 3 folgt eine Beschreibung der Erzeugung von Konstituentenhypothesen unter Einsatz der extrahierten Grammatik. Aus der Grammatik wird durch den Compiler–Generator BISON ein Parser erzeugt, mit dessen Hilfe Wortketten syntaktisch korrekt erweitert werden können. Im Abschnitt 3.1 wird die Vorgehensweise des von BISON erzeugten $LALR(1)$–Parsers erläutert und wie dieser modifiziert werden muß, um bei der Generierung von Konstituentenhypothesen sinnvoll eingesetzt werden zu können. Im Abschnitt 3.2 folgt eine Beschreibung der automatischen Generierung von Konstituentenhypothesen. Kapitel 4 faßt die praktischen Ergebnisse der vorgestellten Algorithmen zusammen. Der Bericht schließt mit einer Zusammenfassung.

Kapitel 2

Automatische Extraktion eines Sprachmodells in Form einer kontextfreien Grammatik aus dem ERNEST–Netzwerk des Sprachverarbeitungssystems EVAR

Das linguistische Wissen des Sprachverarbeitungssystems EVAR ist in einem semantischen Netz abgelegt, das im ERNEST–Formalismus repräsentiert ist. Im Abschnitt 2.1 wird dieser Netzwerkformalismus näher beschrieben. Die Informationen, die in den ERNEST–Datenstrukturen abgelegt sind, werden in den Formalismus einer kontextfreien Grammatik übertragen. Diese Aufgabe ist auf den Teil der Wissensbasis von EVAR beschränkt, der Zeitangaben modelliert. Der betreffende Ausschnitt des ERNEST–Netzes wird im Abschnitt 2.2 vorgestellt. Abschnitt 2.3 schildert anschließend die Einzelheiten der Grammatikerzeugung.

2.1 Der ERNEST–Netzwerkformalismus

In ERNEST werden semantische Netze repräsentiert. Semantische Netze wurden Ende der 60er Jahre als ein grobes Modell des menschlichen Gedächtnisses eingeführt [Quill68]. Informationen über allgemeine Begriffe, Objekte, Ereignisse oder Sachverhalte werden dabei in Knoten und Beziehungen zwischen diesen Begriffen in Kanten repräsentiert. Im Sprachverarbeitungssystem EVAR benötigt man unter anderem Wissen über syntaktische Begriffe, wie zum Beispiel Nomen, Nominalgruppe und so weiter. Zur Darstellung solcher abstrakter Begriffe dient in ERNEST das **Konzept**. Ein Konzept stellt einen Knoten im semantischen Netz dar. Zum Beispiel werden im Konzept SY_TYP_DAT aus dem hier betrachteten EVAR–Sprachnetz Eigenschaften einer allgemeinen Datumsangabe zusammengefaßt. Zwischen diesen Konzepten gibt es Beziehungen, die durch Kanten repräsentiert werden. Eine Tagesangabe ist beispielsweise Bestandteil einer Datumsangabe. Daher ist das Konzept SY_TYP_DAT mit dem Konzept SY_ZEIT_TAG, das einen Wochentag modelliert, durch eine Bestandteilskante verbunden. Neben dem Kantentyp Bestandteil gibt es noch die Kantentypen Spezialisierung und Konkretisierung. Zum Beispiel ist eine Datumsangabe eine spezielle Zeitkonstituente. SY_ZEITKONST repräsentiert Konstituenten für Zeitausdrücke und ist daher mit SY_TYP_DAT durch eine Spezialisierungskante verbunden. Konkretisierungskanten dienen dazu, Begriffe der fünf verschiedenen Abstraktionsebenen *Hypothesen*, *Syntax*, *Semantik*, *Pragmatik* und *Dialog* miteinander zu verbinden. In diesen Abstraktionsebenen ist in EVAR das linguistische Wissen hierarchisch aufgebaut. Die unterste Ebene, die Hypothesenebene enthält Konzepte, die Worthypothesen beschreiben. Die Syntaxebene enthält Wissen darüber, unter welchen Bedingungen ein Wortkette grammatikalisch korrekt ist. Die Semantik der deutschen Sprache, also Wissen über die Bedeutung von Wortketten, ist in der Semantikebene modelliert. Die Pragmatikebene schließlich enthält Begriffe aus dem Anwendungsbereich *Intercity–Auskunft* wie 'Ankunftsort' oder 'Abfahrtszeit'. Den Abschluß dieser Abstraktionshierarchie bildet die Dialogebene, die Interpretationen der Wortketten gemäß dem Dialogverlauf beinhaltet, zum Beispiel

Name	→ Text
Spezialisierung	→ Liste von Kanten zu **Konzepten**
Bestandteil	→ Liste von **Kantenbeschreibungen**
Konkretisierung	→ Liste von **Kantenbeschreibungen**
Modalität	→ Liste von **Modalitätsbeschreibungen**
Attribut	→ Liste von **Attributbeschreibungen**
Bewertung	→ **Funktionsbeschreibung**
⋮	→ ...

Abbildung 2.1: Datenstruktur **Konzept** der ERNEST–Wissensbasis

Rolle	→ Text
Zielknoten	→ Liste von **Konzepten** [XOR-Liste]
Kantenanzahl	→ 2 Integer [min,max]
Bewertung	→ **Funktionsbeschreibung**
Präferenz	→ Liste von **Bereichen**
⋮	→ ...

Abbildung 2.2: Datenstruktur **Kantenbeschreibung** der ERNEST–Wissensbasis

eine einleitende Grußformel zu Beginn eines Gesprächs.

2.1.1 Datenstrukturen der Wissensbasis

Im ERNEST–Formalismus werden die Konzepte durch folgende Datenstrukturen beschrieben. Es wird hier nur das behandelt, was zum Verständnis dieses Berichts nötig ist.

Die Datenstruktur eines **Konzepts** (siehe Abb. 2.1) enthält unter anderem den Konzeptnamen, Listen von Kantenbeschreibungen für die verschiedenen Kantentypen, eine Liste von Modalitätsbeschreibungen und Attribute. Mit Hilfe der Attribute werden die Eigenschaften des modellierten Konzepts beschrieben.

Kanten werden in einer eigenen Datenstruktur **Kantenbeschreibung** repräsentiert (siehe Abb. 2.2). Jeder Kante wird zur Identifikation ein eindeutiger Rollenname zugeordnet. Eine Liste von Zielknoten enthält Konzepte, auf die die Kante jeweils verweist. Es werden also mehrere Konzepte, auf die die gleiche Kante verweist, zusammengefaßt, um eine kompakte Wissensbasis zu erhalten. Der Eintrag *Kantenanzahl* gibt an, wie oft ein Konzept mindestens oder höchstens als Zielkonzept auftreten darf.

Die Datenstruktur **Konzept** enthält weiterhin eine Liste von Modalitätsbeschreibungen (siehe Abb. 2.3). In einer Modalität werden zulässige Gruppierungen von Bestandteilen und Konkretisierungen spezifiziert. Dabei können die entsprechenden Kanten als *obligatorisch*, *optional* oder *inhärent* gekenneichnet werden. *Inhärent* bedeutet, daß das Konzept, auf das die inhärente Kante verweist, obligatorisch für die Gruppierung ist, aber aus verschiedenen Gründen im Sprachsignal nicht gefunden werden kann. Zum Beispiel kann im Lauf eines Dialogs die Frage 'Und wann geht der nächste?' gestellt werden, die das Nomen 'Zug' nicht enthält. Aber wenn in einer vorhergehenden Äußerung dieses Nomen bereits gefunden wurde, kann darauf verwiesen werden.

Zeitliche Lagerelationen innerhalb einer Gruppierung von Bestandteilen und Konkretisierungen werden in der sogenannten *Adjazenz* der Modalität beschrieben. Das Konzept SY_ZEIT_SES zum Beispiel, das eine Sprechereinstellung repräsentiert, enthält die Bestandteilskanten mit den Rollen *nukleus* und *erst_schon*. Die Kante *nukleus* verweist auf das Konzept SY_ADV, das das Wort 'noch' repräsentiert. *erst_schon* hat ebenfalls als Zielkonzept SY_ADV, das in diesem Fall aber die Wörter 'erst' und 'schon' modelliert (siehe Abb. 2.4). SY_ZEIT_SES enthält folgende Modalitätsbeschreibung:

Obligatorisch	→	Liste von *Rollen*
Optional	→	Liste von *Rollen*
Inhärent	→	Liste von *Rollen*
Adjazenz	→	**Adjazenz**

Abbildung 2.3: Datenstruktur **Modalitätsbeschreibung** der ERNEST–Wissensbasis

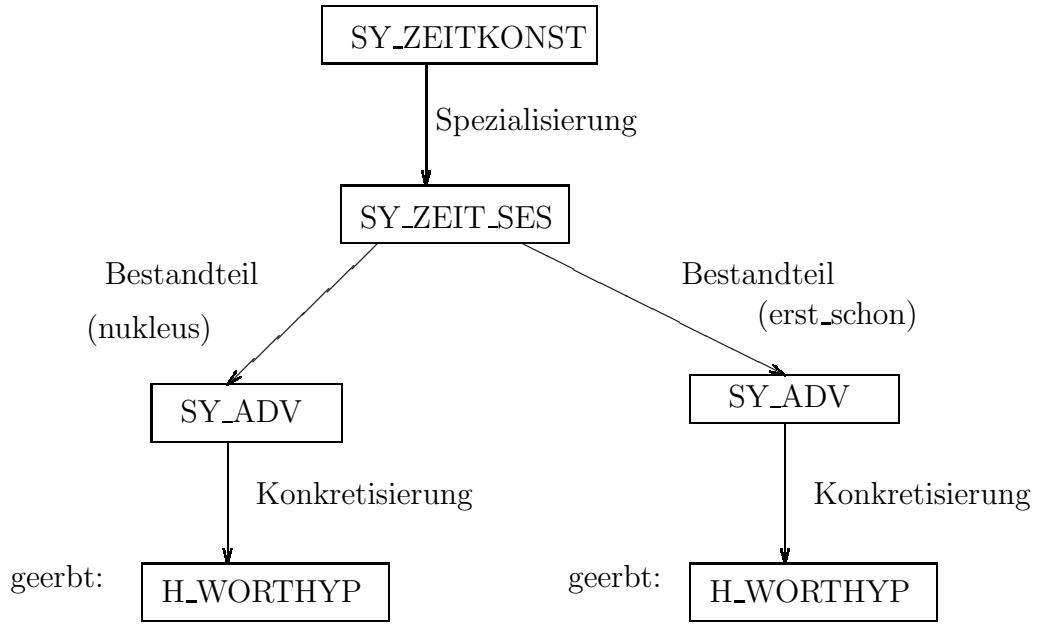


Abbildung 2.4: Ausschnitt aus dem ERNEST–Sprachnetz in EVAR

obligatorisch:	nukleus
optioal:	erst_schon
adjazenz:	01 00

Die Adjazenz enthält die sogenannte Adjazenzmatrix, deren i-ter Zeile und Spalte das i-te Element der Modalität zugeordnet ist. Die erste Zeile und Spalte in diesem Beispiel entspricht also der Kante *nukleus*. Ein Eintrag 1 in der i-ten Zeile und k-ten Spalte bedeutet, daß das k-te Element direkt vor dem i-ten Element stehen darf. Ein Eintrag $n > 1$ in der Diagonale gibt an, daß das entsprechende Element maximal n mal aufeinander folgen darf. Bei den in EVAR modellierten Zeitangaben sind die Einträge der Adjazenzmatrizen auf die Werte 0 und 1 beschränkt. Höhere Diagonalwerte werden im weiteren daher nicht weiter betrachtet. Falls in einem Konzept explizit keine Modalität angegeben ist, so gilt die Konvention, daß alle Bestandteile und Konkretisierungen eine Modalität bilden, in der diese Kanten als obligatorisch gekennzeichnet sind. Im obigen Beispiel besagt die Modalitätsbeschreibung, daß nur die Wortfolgen 'erst noch', 'schon noch' und 'noch' zulässig sind, da die Kante *erst_schon* als optionale Kante nicht auftreten muß. Nähere Angaben zu den Datenstrukturen von ERNEST finden sich in [Manu90], [Kumm90] und [Sage90].

2.1.2 Die Analysekontrolle

Ziel einer Analyse ist es, ein Sprachsignal mit den Begriffen des Problemkreises zu interpretieren. Dazu werden Signalausschnitte mit Konzepten und damit mit deren Bedeutung verbunden [Kumm90]. Diese Verbindung erfolgt über eine Instanz, wobei eine Instanz genau einem Konzept zugeordnet ist. Da die Interpretation eines Signalausschnitts meistens zu einer Einschränkung der zulässigen Interpretationen

für den restlichen Teil des Signals führt, wurden modifizierte Konzepte eingeführt, mit deren Hilfe diese Einschränkungen dargestellt werden können. Ein modifiziertes Konzept repräsentiert Wissen, das an eine konkrete Analysesituation angepaßt wurde. Zur Nutzung des in den Konzepten abgelegten Wissens sind in ERNEST mehrere anwendungsunabhängige Inferenzregeln definiert, mit deren Hilfe Instanzen und modifizierte Konzepte erzeugt werden können [Kumm90]:

Regel 1 und Regel 2: Generierung von Instanzen:
Mit Hilfe dieser Regeln wird zu einem Konzept oder einem modifizierten Konzept eine Instanz erzeugt, falls für alle Bestandteile und Konkretisierungen, die in einer Modalität des Konzepts als obligatorisch gekennzeichnet sind, bereits Instanzen existieren. Ebenso müssen für alle inhärenten Bestandteile und Konkretisierungen Präferenzlisten existieren. In die neu erzeugte Instanz werden die berechneten Werte der Attribute und Bewertungen eingetragen. Die verwendete Modalität wird ebenfalls vermerkt.

Regel 3: Erweiterung von Instanzen:
Wenn Instanzen für Bestandteile und Konkretisierungen einer Instanz existieren, die in deren Modalität als optional gekennzeichnet sind, werden die Kanten dieser Instanz entsprechend erweitert.

Regel 4: Datengetriebene Generierung modifizierter Konzepte:
Durch die Erzeugung von Instanzen können zulässige Instanzen zu anderen Konzepten beschränkt werden. Mit Hilfe von Regel 4 können daher modifizierte Konzepte gebildet werden, falls Instanzen oder modifizierte Konzepte für obligatorische oder optionale Bestandteils- beziehungsweise Konkretisierungskanten existieren, oder wenn für inhärente Kanten Präferenzlisten existieren. In diese modifizierten Konzepte werden die erforderlichen Beschränkungen von Attributen, zum Beispiel von Wertebereichen, eingetragen.

Regel 5: Modellgetriebene Generierung modifizierter Konzepte:
Vorerwartungen des Modells und datengetriebene Restriktionen werden mit Regel 5 an Konzepte tieferer Hierarchiestufen weitergereicht. Durch diese Regel werden modifizierte Konzepte für alle Bestandteile und Konkretisierungen einer Modalitätsbeschreibung von bereits existierenden Instanzen oder modifizierten Konzepten erzeugt.

Regel 6: Initiale Schätzung von Analysezielen:
Zu Beginn der Analyse kann ein Analyseziel in Form von Zielkonzepten angegeben werden, das heißt, die Analysekontrolle versucht, diese Konzepte zu instantiieren. Durch Regel 6 werden zu diesen Zielkonzepten, die auch während der Analyse neu berechnet werden können, modifizierte Konzepte gebildet.

Im allgemeinen ist in semantischen Netzen eine natürliche Inferenzregel, die Vererbung, definiert. Dieser Vererbungsmechanismus ist anwendbar auf Konzepte, die über Spezialisierungskanten miteinander verbunden sind. Folgende Informationen werden dabei vererbt: alle Bestandteile und Konkretisierungen, alle Attribute und Relationen. Es wird dadurch die Erzeugung und Verwaltung einer kompakten Wissensbasis ermöglicht. Das Konzept SY_ADV, das ein Adverb repräsentiert, zum Beispiel ist eine Spezialisierung des Konzepts SY_WORT_KLASS, das eine Wortklasse mit semantischer und pragmatischer Klasse modelliert. Die Konkretisierung H_WORTHYP wird daher an SY_ADV vererbt (siehe Abb. 2.4).

Das Ziel der Wissensnutzung in ERNEST besteht in der Erzeugung einer Instanz, die eine vollständige Beschreibung des Sensorsignals repräsentiert. Um dieses Ziel zu erreichen, stellt der ERNEST–Netzwerkformalismus einen problemunabhängigen Kontrollalgorithmus zur Verfügung. Dieser Bearbeitungsvorschrift liegt der A^*–Algorithmus (siehe Kapitel 3.2.1 und [Nils71]) zugrunde. Durch die Anwendung der oben beschriebenen Inferenzregeln werden Mengen von konkurrierenden Instanzen und modifizierten Konzepten erzeugt, die in einem Suchgraphen gegliedert werden. Ein Pfad in diesem Graphen vom Wurzelknoten zu einem Endknoten entspricht einer möglichen Interpretation des gesamten Signals auf verschiedenen Abstraktionsebenen. Während der Analyse wird ein solcher Suchgraph durch Anwendung der sechs Inferenzregeln aufgebaut, die zu jedem Knoten die zulässigen Nachfolgeknoten bestimmen. Die dabei entstehenden Suchbaumknoten erhalten eine anwendungsabhängige Bewertung, mit deren Hilfe der bestbewertete Lösungspfad, also die bestmögliche Interpretation gefunden wird.

Zwei grundlegende Verarbeitungsstrategien können verfolgt werden. Bei der datengetriebenen Analyse werden zuerst die sogenannten initialisierenden Konzepte, Konzepte aus der Hypothesenebene, instantiiert. Durch Anwendung der Inferenzregeln können dadurch Konzepte aus höheren Ebenen instantiiert werden, bis eine Instanz des Analyseziels erreicht ist. Bei der modellgetriebenen beziehungsweise erwartungsgesteuerten Analyse werden, von einem Zielkonzept ausgehend, modifizierte Konzepte erzeugt. Der Vorgang wird wiederholt, bis die initialisierenden Konzepte erreicht worden sind, die dann instantiiert werden. Im Sprachverarbeitungssystem EVAR wird eine Kombination von beiden Strategien verwendet.

Der problemunabhängige Kontrollalgorithmus in ERNEST verfolgt folgenden Arbeitsablauf: Eingabeparameter ist eine nichtleere Liste vorläufiger Zielkonzepte. Es besteht die Möglichkeit, daß nach der erwartungsgesteuerten Instantiierung dieser Konzepte neue Zielkonzepte auf höherer Ebene geschätzt werden. Für die Bearbeitung ist eine Menge OFFEN erforderlich, die alle noch nicht vollständig bearbeiteten Suchbaumknoten enthält. Zu Beginn der Analyse wird eine anwendungsabhängige Initialisierung durchgeführt. Im Fall von EVAR werden konkurrierende Worthypothesen erzeugt und Konzepte aus der Hypothesenebene entsprechend instantiiert. Für alle Zielkonzepte werden modifizierte Konzepte kreiert, die in verschiedenen Suchbaumknoten abgelegt werden, da sie konkurrierende Zielinterpretationen darstellen. Diese Suchbaumknoten werden bewertet und in die Menge OFFEN eingetragen. Die Menge OFFEN wird nun solange bearbeitet, bis sie keine Knoten mehr enthält, also alle vollständig bearbeitet worden sind. Der jeweils bestbewertete Knoten wird aus OFFEN entfernt und danach untersucht, ob ein vom ERNEST–Benutzer definiertes Abbruchkriterium für den Kontrollalgorithmus erfüllt ist oder nicht. Die Analyse kann zum Beispiel dann beendet sein, wenn ein vorher festgelegtes Zielkonzept instantiiert wurde. Falls dieses Ziel noch nicht erreicht ist, wird der Knoten expandiert. Ein Suchbaumknoten enthält ein Netz, bestehend aus Instanzen und modifizierten Konzepten, das eine Teilinterpretation des Sensorsignals repräsentiert. Auf die Instanzen und modifizierten Konzepte werden die Inferenzregeln angewendet und das Netz entsprechend geändert beziehungsweise erweitert. Die neu entstehenden Netze, die möglicherweise konkurrierende Interpretationen darstellen, werden als neue Nachfolgerknoten des betrachteten Suchbaumknotens in den Suchbaum eingehängt. Diese Knoten werden ebenfalls bewertet und in die Menge OFFEN eingetragen. Nach Beendigung des gesamten Algorithmus stellt der bestbewertete Pfad in dem erzeugten Suchbaum die bestmögliche Interpretation des Sensorsignals dar. Die Bewertung eines Pfades ergibt sich aus der Summe der Bewertungen der einzelnen Knoten, die der Pfad durchläuft. Nähere Einzelheiten zu diesem Kontrollalgorithmus sind in [Kumm90] nachzulesen.

2.2 Die Zeitangaben des Sprachverarbeitungssystems EVAR

Im Anwendungsbereich 'Intercity–Zugauskunft' des Sprachverarbeitungssystems EVAR spielen Zeitangaben eine große Rolle. Daher sind in einem Teil der Wissensbasis Zeitangaben, wie zum Beispiel

> morgen vor sieben Uhr abends, heute, ungefähr um elf Uhr, morgen früh, irgendwann heute nachmittag, in drei Wochen am Donnerstag nachmittag

modelliert. Da die Aufgabe der Grammatikextraktion hier auf den Teil der Zeitangaben beschränkt ist, wird nur der entsprechende Ausschnitt aus dem ERNEST–Netzwerk betrachtet.

Die Kriterien für die Zulässigkeit von Wortketten, die Zeitangaben darstellen, wurden im Rahmen von ([Diez88]) in Form von ATN–Grammatiken (augmented transition network) bestimmt. Diese Übergangsnetzwerke wurden anschließend in die ERNEST–Wissensbasis von EVAR übertragen.

Die Zeitangaben sind in Kategorien, wie zum Beispiel *Datum*, *Uhrzeit* oder *Tendenz*, unterteilt. Jede Kategorie wird durch ein Konzept im ERNEST–Netz modelliert. Die Namen dieser Konzepte beginnen mit dem Präfix 'SY_ZEIT_'. SY_ZEIT_DAU zum Beispiel repräsentiert verschiedene Angaben einer Zeitdauer wie '*besonders kurz*', '*so lang wie möglich*' usw. Die Kategorien können zu Ketten kombiniert werden. '*am zehnten Oktober um drei Uhr*' beispielsweise ist eine Kombination eines Datums und einer Uhrzeitangabe. Die Namen der Konzepte, die solche Ketten darstellen, beginnen mit 'SY_TYP_'. Beispielsweise modelliert SY_TYP_SPANNE Wortketten wie '*alle dreißig Minuten möglichst früh*', eine Kombination einer Zeiteinheit und einer Tendenz. Das Konzept SY_ZEITKETTE schließlich repräsentiert die gesamte Zeitangabe einer Äußerung.

Wortketten, die Zeitangaben darstellen, sind aufgebaut aus Nomen, Präpositionen, Zahlwörtern usw. Diese Wortarten sind also Bestandteile der oben genannten Zeitbegriffe. Jede Wortart ist ebenfalls

durch ein Konzept repräsentiert. Durch SY_PRAEP zum Beispiel werden Präpositionen dargestellt. Die Wortarten selbst werden durch Wörter konkretisiert. Die entsprechenden Konzepte enthalten daher als Konkretisierung das Konzept H_WORTHYP aus der Hypothesenebene.

Die zeitlichen Lagerelationen der Wortarten sind in den Modalitätsbeschreibungen der Konzepte, die Zeitbegriffe repräsentieren, definiert. Zum Beispiel legt die Modalitätsbeschreibung des Konzepts SY_ZEIT_DAU fest, daß nicht jede Kombination der Wortarten 'Adverb', 'unflektiertes Adjektiv' und 'Vergleichspartikel' möglich ist. '*möglich so kurz wie*' ist danach nicht korrekt, '*so kurz wie möglich*' jedoch ist erlaubt.

Im Anhang A sind alle Konzepte der linguistischen Wissensbasis, die Zeitangaben betreffen, aufgeführt und kurz beschrieben.

2.3 Automatische Extraktion einer kontextfreien Grammatik für die Zeitangaben

Um in der akustisch–phonetischen Dekodierung möglichst frühzeitig wissensbezogene Restriktionen einsetzen zu können, ist ein Sprachmodell erforderlich, das syntaktische, semantische, pragmatische und dialogspezifische Einschränkungen aus der Wissensbasis repräsentiert. Das gesuchte Sprachmodell wird hier in Form einer kontextfreien Grammatik dargestellt. Die erforderliche Darstellung dieser Grammatik wird in Abschnitt 2.3.1 dargelegt. Die Daten aus der Wissensbasis, also aus dem ERNEST–Netz, die nötig sind, um die gesuchte Grammatik erzeugen zu können, werden im Abschnitt 2.3.2 aufgeführt. Der ganze Algorithmus zum Erzeugen der kontextfreien Grammatik schließlich wird im Abschnitt 2.3.3 erläutert. In Kapitel 3 folgt anschließend eine Beschreibung, wie diese Grammatik bei der Analyse des Sprachsignals eingesetzt wird.

2.3.1 Darstellung einer kontextfreien Grammatik

Eine **kontextfreie Grammatik** G ist definiert durch ein Quadrupel $G = (N, T, P, S)$, wobei

- N eine Menge von Nichtterminalsymbolen,
- T eine Menge von Terminalsymbolen,
- P eine endliche, nichtleere Menge von Produktionen und
- S ein spezielles Nichtterminalsymbol, das Startsymbol, sind.

Die Menge P der Produktionen ist eine Teilmenge von $N \times V^*$, wobei $V = N \cup T$ die Menge aller Grammatiksymbole ist. Eine Produktion oder Regel der Grammatik wird als $x \longrightarrow y$ geschrieben. $x \in N$ ist die linke und $y \in V^*$ die rechte Seite der Regel. Sätze der so definierten Grammatiksprache werden ausgehend vom Startsymbol durch iterierte Anwendung der Grammatikregeln erzeugt. Nichtterminalsymbole, die die linke Seite einer Regel bilden, werden solange durch die entsprechende rechte Regelseite ersetzt, bis das erzeugte Wort nur noch aus Terminalsymbolen besteht. Alle Sätze, die auf diese Weise aus dem Startsymbol herleitbar sind, bilden die Sprache der Grammatik.

Ein Programm, das für einen vorgegebenen Satz entscheidet, ob er in der Sprache einer Grammatik enthalten ist oder nicht, nennt man *Parser*. Ein Parser kann automatisch aus einer Grammatik erzeugt werden. Unter dem Betriebssystem UNIX wird der Compilergenerator BISON, eine Weiterentwicklung des YACC [John74], zur Verfügung gestellt. Bei der Analyse des Sprachsignals wird der von BISON generierte Parser eingesetzt (siehe Kapitel 3). Die kontextfreie Grammatik, die aus dem ERNEST–Netz extrahiert wird, dient als Eingabe des BISON–Parsers. Daher muß sie in dem von BISON erwarteten Format dargestellt sein.

Um die Struktur der Zeitangaben, die im ERNEST–Netz enthalten ist, in einen Grammatikformalismus zu übertragen, werden die Grammatiksymbole wie folgt festgelegt: Als Nichtterminalsymbole werden Konzeptnamen gewählt, denen zum Zweck der Unterscheidung (siehe Abschnitt 2.3.2) Ziffern angehängt werden. *SY_ADV_1* zum Beispiel ist ein vollständiges Nichtterminalsymbol. Das Startsymbol der kontexfreien Grammatik wird aus dem Konzept gebildet, das die Wurzel des betrachteten Netzes aus der Wissensbasis darstellt. Da hier nur Zeitangaben betrachtet werden, ist SY_ZEITKETTE_0 ein geeignetes Startsymbol. Terminalsymbole sind diejenigen Symbole, aus denen die Sätze der durch

die Grammatik festgelegten Sprache ausschließlich bestehen. Im Sprachverarbeitungssystem EVAR werden die einzelnen Wörter der deutschen Sprache durch Wortnummern repräsentiert. Hier sind nur die Wörter von Bedeutung, die im Anwendungsbereich 'Intercity–Auskunft' eine Rolle spielen. Die Terminalsymbole werden aus diesen Wortnummern gebildet. Der BISON–Parser erwartet als Eingabe Grammatiksymbole, die von beliebiger Länge sein dürfen und aus Buchstaben, Punkten '.', Unterstrichen '_' und Ziffern aufgebaut sein müssen. Das erste Zeichen allerdings darf keine Ziffer sein (siehe [John74]). Aus diesem Grund werden, um gültige Terminalsymbole zu erhalten, den Wortnumern Punkte vorangestellt. '.516' ist beispielsweise ein Terminalsymbol und repräsentiert das Wort 'schon'.

Eine Grammatikregel $A \longrightarrow y$ wird wie folgt dargestellt, wobei die linke Regelseite A ein Nichtterminalsymbol und die rechte Seite y eine Folge von Grammatiksymbolen ist, die auch leer sein kann:

$$A \ : \quad y \ ;$$

Mehrere Grammatikregeln $A \longrightarrow y_1$, $A \longrightarrow y_2 \ldots$ mit derselben linken Seite können zu einer zusammengesetzten Regel vereinigt werden:

$$\begin{array}{lll} A & : & y_1 \\ & | & y_2 \\ & & \vdots \\ & ; & \end{array}$$

Terminalsymbole müssen zu Beginn der Grammatikdefinition mit

$$\%token \qquad \text{Terminalsymbol}$$

vereinbart werden. Nach dem Trennsymbol '%%' folgen dann die Definitionen der Regeln. Es ist dabei festgelegt, daß das Nichtterminalsymbol auf der linken Seite der ersten Regel das Startsymbol bildet.

Beispiel: Folgende Grammatik ist im BISON–Format dargestellt. Die Wortnummern 3851, 375 und 378 repräsentieren die Wörter *erst*, *noch* und *schon*. Die Sprache der Grammatik besteht aus den Wortfolgen '*noch*', '*erst noch*' und '*schon noch*':

```
%token .3851
%token .375
%token .378

%%

SY_ZEIT_SES_0   : SY_ADV_1
                | SY_ADV_3 SY_ADV_1
                ;
SY_ADV_1        : H_WORTHYP_2
                ;
H_WORTHYP_2     : .375
                ;
SY_ADV_3        : H_WORTHYP_4
                ;
H_WORTHYP_4     : .3851
                | .378
                ;
```

2.3.2 Daten des ERNEST–Netzes, die für die Extraktion der kontextfreien Grammatik nötig sind

Um eine aussagekräftige Grammatik bilden zu können, müssen verschiedene Informationen verarbeitet werden, die durch die Datenstrukturen der ERNEST–Wissensbasis gegeben sind. Interessant sind hier die Kanten und die Modalitätsbeschreibungen der Konzepte. Der Teil der EVAR–Wissensbasis, der die Zeitangaben modelliert, enthält nur Bestandteils- und Konkretisierungskanten. Spezialisierungskanten werden daher nicht weiter betrachtet.

Läßt sich der Begriff, der einem Konzept A zugrundeliegt, in ein oder mehrere Bestandteile B_1, $B_2 \ldots$ zergliedern, sind die entsprechenden Konzepte mit A durch Bestandteilskanten verbunden. Entsprechendes gilt für Konkretisierungen. Diese Tatsache läßt sich einfach in den Formalismus einer

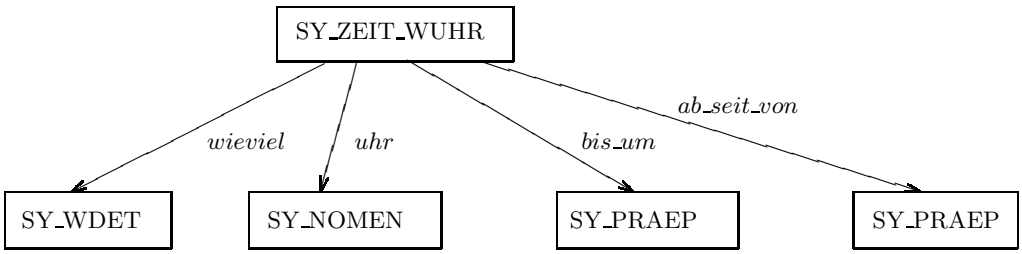


Abbildung 2.5: Ausschnitt aus der ERNEST–Wissensbasis

kontextfreien Grammatik umsetzen: Die Regel $A \longrightarrow B_1 B_2 \ldots B_n$ drückt aus, das sich A durch die Folge von Bestandteilen beziehungsweise Konkretisierungen $B_1 B_2 \ldots B_n$ ersetzen läßt. Wie diese Folge aufgebaut ist, das heißt welche Bestandteile und Konkretisierungen enthalten sind und in welcher Reihenfolge sie erscheinen, ist nicht beliebig. Es können auch mehrere verschiedene Folgen möglich sein, wodurch mehrere Regeln mit A als linker Seite erforderlich sind. Diese Gegebenheiten sind durch die Modalitätsbeschreibungen des übergeordneten Konzepts A festgelegt.

Beispiel: Das Konzept SY_ZEIT_WUHR, das eine Frage nach einer Uhrzeit repräsentiert, ist durch die Bestandteilskanten '*wieviel*', '*uhr*', '*bis_um*' und '*ab_seit_von*' mit den Konzepten SY_WDET (Fragedeterminans), SY_NOMEN (Nomen) und SY_PRAEP (Präposition) verbunden (siehe Abbildung 2.5). SY_ZEIT_WUHR enthält folgende Modalitätsbeschreibung:

obligatorisch:	*wieviel*, *uhr*
optional:	*bis_um*, *ab_seit_von*
Adjazenz:	0 0 1 1
	1 0 0 0
	0 0 0 0
	0 0 0 0

Daraus ergibt sich die zusammengesetzte Grammatikregel:

$$
\begin{array}{lcl}
SY\_ZEIT\_WUHR\_1 & : & SY\_WDET\_2 \;\; SY\_NOMEN\_3 \\
& | & SY\_PRAEP\_4 \;\; SY\_WDET\_2 \;\; SY\_NOMEN\_3 \\
& | & SY\_PRAEP\_5 \;\; SY\_WDET\_2 \;\; SY\_NOMEN\_3 \\
& ; &
\end{array}
$$

Aus diesem Beispiel wird ersichtlich, weshalb die an die Konzeptnamen angefügten Ziffern notwendig sind. Das Konzept SY_PRAEP ist durch zwei verschiedene Kanten mit SY_ZEIT_WUHR verbunden. Es repräsentiert in jedem der beiden Fälle eine andere Menge von Wörtern, die durch die jeweilige Kantenbeschreibung festgelegt ist. Es muß somit eine Unterscheidung getroffen werden. Durch die Modalitätsbeschreibung sind drei verschiedene Folgen von Grammatiksymbolen auf der rechten Regelseite möglich. Jede Folge enthält die Symbole SY_WDET_2 und SY_NOMEN_3, da sowohl das Konzept SY_WDET als auch das Konzept SY_NOMEN mit SY_ZEIT_WUHR durch obligatorische Kanten (*wieviel*, *uhr*) verbunden sind. Die Kanten *bis_um* und *ab_seit_von* sind als optional gekennzeichnet. Dadurch müssen auch Regeln erzeugt werden, auf deren rechten Regelseite die Symbole SY_PRAEP_4 beziehungsweise SY_PRAEP_5 nicht vorkommen. Dasselbe würde gelten, wären diese Kanten als inhärent markiert. Die Reihenfolge der Grammatiksymbole schließlich wird durch die Adjazenzmatrix festgelegt. Sie sagt hier aus, daß die Kanten *wieviel* nach *bis_um*, *wieviel* nach *ab_seit_von* und *uhr* nach *wieviel* folgen dürfen.

Weitere Daten, die für die Erzeugung einer kontextfreien Grammatik erforderlich sind und die die syntaktischen, semantischen, pragmatischen und dialogspezifischen Beschränkungen der ERNEST–Wissensbasis wiedergeben, liefern die Kantenrestriktionen im ERNEST–Netz. Im Eintrag '*bewertung*' der Kantenbeschreibungen können Restriktionen vermerkt werden, die die Zulässigkeit der Kanten

einschränken. Zum Beispiel enthält die Kantenrestriktion der Bestandteilskante *uhr* des Konzepts SY_ZEIT_WUHR die Bedingung, daß das Zielkonzept SY_NOMEN nur das Wort 'Uhr' repräsentieren darf. Diese Restriktion könnte durch die Grammatikregel

$$SY\_ZEIT\_WUHR\_1 \ : \quad .124 \ ;$$

ausgedrückt werden. 124 ist die das Nomen 'Uhr' repräsentierende Wortnummer. Im Eintrag '*bewertung*' der Kantenrestriktionen sind aber die Vorschriften, wie der Gültigkeitsbereich des jeweiligen Zielkonzepts eingeschränkt wird, in Form von C–Programmprozeduren abgelegt. Das heißt, die hier interessierenden Informationen sind im Programmcode dieser Prozeduren verborgen.

Ein weiteres Problem bei der Grammatikgenerierung stellt der Vererbungsmechanismus von ERNEST dar. Die über Spezialisierungskanten vererbten Strukturen, die erst in der Vorbereitungsphase der Analyse berechnet werden, müssen bei der Erzeugung der Grammatikregeln berücksichtigt werden.

Alle Daten, die nötig sind, um das Wissen im ERNEST–Netz in eine kontexfreie Grammatik zu übertragen, können also nicht direkt aus der Wissensbasis herausgelesen werden. Das Problem wird wie folgt gelöst: Durch die modellgetriebene Generierung modifizierter Konzepte (Regel 5) werden während des Ablaufs der Analysekontrolle die Kantenrestriktionen ausgewertet und die entsprechenden Einschränkungen in die modifizierten Konzepte eingetragen. Zum Beispiel wird über die Kante *uhr* von SY_ZEIT_WUHR ein modifiziertes Konzept zu SY_NOMEN kreiert, dessen vererbter Analyseparameter '*wort_nr*' mit der Wortnummer 124 bestimmt wird. Durch wiederholte Anwendung der Regel 5 werden zu Bestandteilen und Konkretisierungen weitere modifizierte Konzepte tieferer Hierarchiestufen generiert, in die die einmal berechneten Einschränkungen übertragen werden. Auf diese Weise werden Kantenrestriktionen höher liegender Konzepte bis in die Hypothesenebene weitergereicht. Beispielsweise wird beim Erzeugen eines modifizierten Konzepts zu H_WORTHYP über die vererbte Konkretisierungskante des bereits erzeugten modifizierten Konzepts SY_NOMEN das Attribut '*anforderung*' von H_WORTHYP mit der Wortnummer 124 berechnet. Das modifizierte Konzept zu H_WORTHYP repräsentiert also das Wort 'Uhr'. Diese Tatsache kann mit folgender Grammatikregel realisiert werden:

$$H\_WORTHYP\_1 \ : \quad .124 \ ;$$

Wenn nun, ausgehend vom Zielkonzept SY_ZEITKETTE, mit Hilfe der Regel 5 ein Netz erzeugt wird, das ausschließlich aus modifizierten Konzepten aufgebaut ist, können aus den modifizierten Konzepten der Hypothesenebene die Wortnummern herausgelesen werden, auf die der entsprechende Gültigkeitsbereich eingeschränkt ist. Um ein solches Netz mit Hilfe des ERNEST–Kontrollalgorithmus zu generieren, werden folgende Modifikationen an diesem Algorithmus vorgenommen:

Da keine Analyse eines Sprachsignals vorgenommen werden soll, werden die initialisierenden Instantiierungen unterdrückt. Der Algorithmus beginnt dann durch Anwendung der Regel 6 mit der Erzeugung eines modifizierten Konzepts zum Zielkonzept. Anschließend wird die Menge OFFEN wie üblich bearbeitet. Der jeweils bestbewertete Knoten wird ausgewählt und bearbeitet. Wenn dieser Knoten kein Zielknoten ist, wird er expandiert. Steht ein modifiziertes Konzept in diesem Knoten zur Instantiierung an, das heißt, es stammt aus der Hypothesenebene, dann wird es übergangen. Dies ist notwendig, da im ursprünglichen Kontrollalgorithmus die Instantiierung der Expansion eines modifizierten Konzepts vorgezogen wird. Durch die eben erwähnte Änderung kommt nur die Regel 5 zur Anwendung, und es werden nur modifizierte Konzepte aber keine Instanzen erzeugt. Der modifizierte Kontrollalgorithmus endet, wenn die Menge OFFEN vollständig bearbeitet, also leer ist. Zielknoten sind in diesem Fall nicht definiert. Es entsteht demnach ein Suchbaum, dessen Blattknoten ausschließlich modifizierte Konzepte enthalten, die nicht mehr weiter bearbeitet werden können oder zur Instantiierung anstehen. Diese modifizierten Konzepte sind durch die entsprechenden Kanten miteinander verbunden und bilden je Blattknoten einen Baum mit dem zum Zielkonzept gehörenden modifizierten Konzept als Wurzel. Die Blätter dieser Bäume werden aus modifizierten Konzepten der Hypothesenebene gebildet, bis zu denen die Kantenrestriktionen der darüberliegenden Konzepte weitergereicht wurden. Zusätzlich werden durch die Vorbereitungsphase der Kontrolle die vererbten Strukturen mit berücksichtigt. Aus den Blattknoten des so generierten Suchbaums können also alle Daten, die für die Extraktion der kontextfreien Grammatik nötig sind, problemlos extrahiert werden.

Beispiel: Die Abbildungen 2.6 bis 2.8 zeigen die Blätter des Suchbaums, der entsteht, wenn der modifizierte Kontrollalgorithmus auf das Konzept SY_ZEIT_DAU angewendet wird.

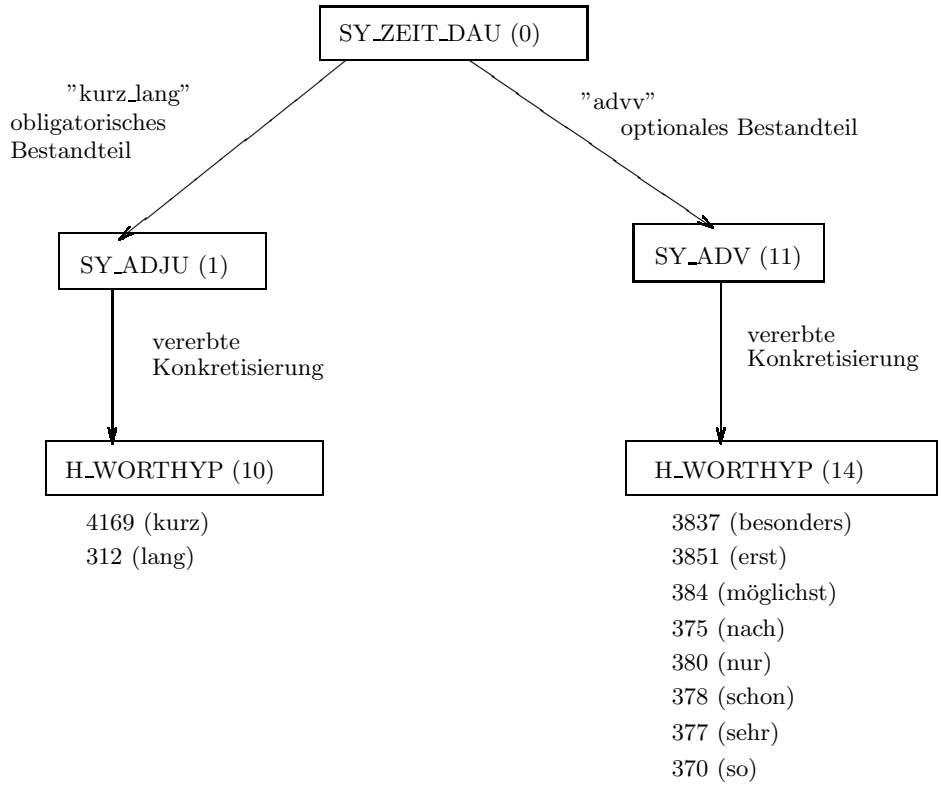


Abbildung 2.6: Erster Blattknoten des Suchbaums, der bei der Analyse des Konzepts SY_ZEIT_DAU generiert wird. Verarbeitet wurde folgende Modalitätsbeschreibung:

beschreibung:	obligatorisch:	*kurz_lang*
	optional:	*advv*
	Adjazenz:	0 1
		0 0

Die modifizierten Konzepte erhalten eindeutige Identifikationen. Diese Identifikationen sind in den Abbildungen 2.6 bis 2.8 hinter den jeweiligen Konzeptnamen in Klammern vermerkt. Sie werden verwendet, um Nichtterminalsymbole zu erzeugen. Dazu werden an die Konzeptnamen die den modifizierten Konzepten zugeordneten Identifikationen hinzugefügt. Unter den modifizierten Konzepten der Hypothesenebene sind die Wortnummern mit den entsprechenden Wörtern aufgelistet, auf die der Gültigkeitsbereich des Konzepts darüber eingeschränkt ist. Aus den Suchbaumblättern läßt sich also folgende Grammatik extrahieren:

```
%token .286
%token .312
%token .368
%token .370
%token .375
%token .377
%token .378
%token .380
%token .384
%token .3837
%token .3851
%token .4169

%%

SY_ZEIT_DAU_0
                : SY_ADV_6 SY_ADJU_2 SY_VGLPAR_8 SY_ADJU_4
                | SY_ADJU_1
```

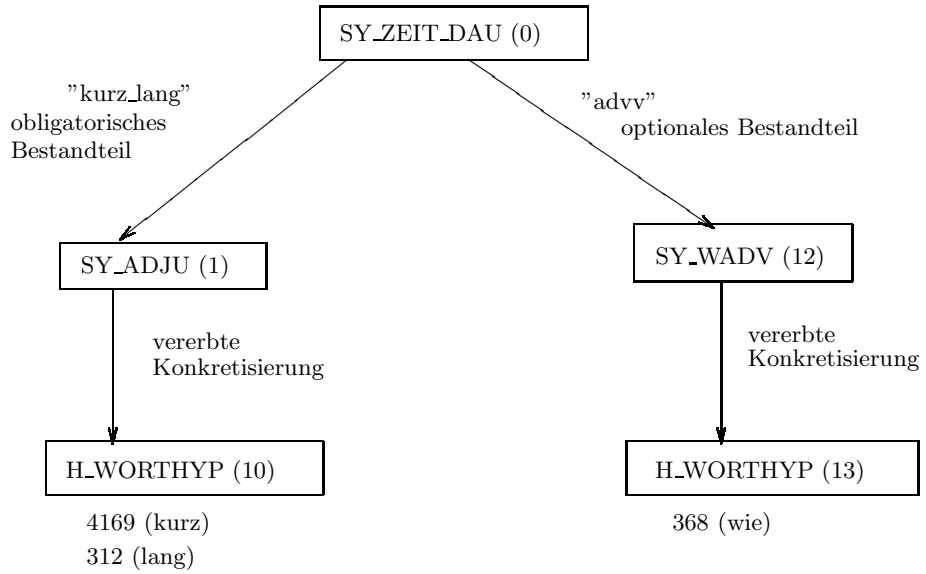


Abbildung 2.7: Zweiter Blattknoten des Suchbaums, der bei der Analyse des Konzepts SY_ZEIT_DAU generiert wird. Verarbeitet wurde folgende Modalitätsbeschreibung:

obligatorisch:	*kurz_lang*
optional:	*advv*
Adjazenz:	0 1
	0 0

```
                | SY_WADV_12 SY_ADJU_1
                | SY_ADV_11 SY_ADJU_1
                ;
SY_ADJU_2
                : H_WORTHYP_3
                ;
H_WORTHYP_3
                : .4169
                | .312
                ;
SY_ADJU_4
                : H_WORTHYP_5
                ;
H_WORTHYP_5
                : .286
                ;
SY_ADV_6
                : H_WORTHYP_7
                ;
H_WORTHYP_7
                : .370
                ;
SY_VGLPAR_8
                : H_WORTHYP_9
                ;
H_WORTHYP_9
                : .368
                ;
SY_ADJU_1
                : H_WORTHYP_10
                ;
H_WORTHYP_10
                : .4169
                | .312
                ;
```

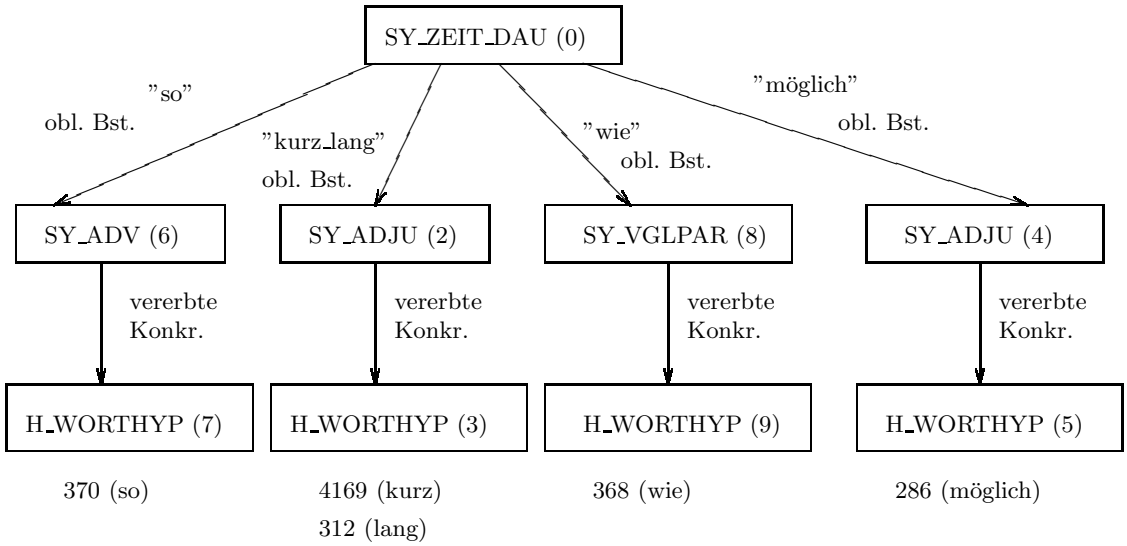


Abbildung 2.8: Dritter Blattknoten des Suchbaums, der bei der Analyse des Konzepts SY_ZEIT_DAU generiert wird. Verarbeitet wurde folgende Modalitätsbeschreibung:

obligatorisch: *so*, *kurz_lang*, *wie*, *möglich*

Adjazenz:

0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0

```
SY_WADV_12
                : H_WORTHYP_13
                ;
H_WORTHYP_13
                : .368
                ;
SY_ADV_11
                : H_WORTHYP_14
                ;
H_WORTHYP_14
                : .3837
                | .3851
                | .384
                | .375
                | .380
                | .378
                | .377
                | .370
                ;
```

2.3.3 Algorithmus zum Extrahieren der Grammatik aus dem ERNEST–Netz

Mit Hilfe des im vorigen Abschnitt beschriebenen modifizierten ERNEST–Kontrollalgorithmus wird ein Suchbaum erzeugt, aus dessen Blattknoten die gesuchte kontextfreie Grammatik extrahiert werden kann. Der Wurzelknoten dieses Suchbaums enthält das modifizierte Zielkonzept. Bei jeder Expansion eines Suchbaumknotens wird ein modifiziertes, noch nicht bearbeitetes Konzept ausgewählt, das nicht zur Instantiierung ansteht. Gemäß Regel 6 werden zu den Bestandteilen und Konkretisierungen jeder einzelnen Modalität dieses Konzepts modifizzierte Konzepte generiert. Jedesmal, wenn ein neues modifiziertes Konzept erzeugt worden ist, wird ein neuer Suchbaumknoten kreiert, der das um dieses

Knotenbewertung	→ **Bewertung**
Vorgängerknoten	→ **Suchbaumknoten**
Nachfolgerknoten	→ Liste von **Suchbaumknoten**
Knoteneintrag des Zielkonzepts	→ **Knoteneintrag**

Abbildung 2.9: Datenstruktur **Suchbaumknoten** der ERNEST–Kontrolle

Nachfolgereintrag	→ Liste von **Knoteneinträgen**
Eintragsinformation	→ **Eintragsinformation**
⋮	→ ...

Abbildung 2.10: Datenstruktur **Knoteneintrag** der ERNEST–Kontrolle

neue modifizierte Konzept erweiterte Netz seines Vorgänger erhält. Besitzt ein Konzept zwei oder mehr Modalitätsbeschreibungen oder verweist eine Kante auf mehrere unterschiedliche Zielkonzepte werden mehrere alternative Suchbaumknoten erzeugt, die alle Nachfolger des ausgewählten Suchbaumknotens bilden. Die Blattknoten des Suchbaums enthalten schließlich Netze, die ausschließlich aus modifizierten Konzepten bestehen und bis in die Hypothesenebene berechnet worden sind. In ihrer Gesamtheit repräsentieren diese Blattknoten sämtliche mögliche Wortketten, die gemäß der EVAR–Wissensbasis Zeitangaben darstellen, wenn als Zielkonzept SY_ZEITKETTE gewählt worden ist.

Der Suchbaum, aus dem nun die nötigen Daten für die Grammatik herausgelesen werden können, ist in folgenden Datenstrukturen abgelegt: Ein Suchbaumknoten (siehe Abbildung 2.9) enthält als Einträge eine Bewertung, einen Vorgängerknoten, eine Liste von Nachfolgerknoten und das Zielkonzept, also die Wurzel des aus modifizierten Konzepten bestehenden Netzes, das der Suchbaumknoten enthält. Die Bewertung ist für den ursprünglichen Kontrollalgorithmus nötig, um die bestbewertete Interpretation des Sprachsignals zu finden. Da in der hier interessierenden Anwendung alle möglichen Expansionen ausgeführt werden müssen, um alle Alternativen zu erhalten, spielt die Bewertung keine Rolle.

Modifizierte Konzepte und Instanzen innerhalb eines Suchbaumknotens sind in Form von Knoteneinträgen abgelegt (siehe Abbildung 2.10). Ein Knoteneintrag enthält unter anderem die Nachfolgeeinträge und eine Datenstruktur, die alle Informationen enthält, die das modifizierte Konzept beziehungsweise die Instanz betreffen. Diese sogenannte Eintragsinformation (siehe Abbildung 2.11) gibt zum Beispiel Auskunft darüber, was für ein Objekt, also ein modifiziertes Konzept oder eine Instanz, beschrieben wird oder welche Modalitätsbeschreibung bei der Generierung der darunterhängenden Bestandteile und Konkretisierungen verwendet wurde.

Für jeden Knoteneintrag aller Suchbaumblattknoten wird nun eine, gegebenenfalls zusammengesetzte, Grammatikregel im BISON–Eingabeformat erzeugt. Der Name des dem Knoteneintrag zugrundeliegenden modifizierten Konzepts und die dazugehörige Identifikation ergeben das Nichtterminalsymbol auf der linken Seite der Regel. Beides wird aus der Eintragsinformation gewonnen. Durch Auswertung der Modalitätsbeschreibung wird die rechte Regelseite erzeugt. Dabei wird ein **Symbolbaum** aufgebaut, um die alternativen Ketten von Grammatiksymbolen für die spätere Ausgabe zwischenzuspeichern.

Ein Symbolbaumknoten enthält ein Grammatiksymbol. Alle Symbole, die direkt nach diesem Grammatiksymbol stehen dürfen, werden in einer Nachfolgerliste abgelegt. Diese Nachfolger repräsentieren verschiedene Alternativen, mit denen das Nichtterminalsymbol zu einer möglichen Kette vervollständigt werden kann. Ein Pfad vom Wurzelknoten des Symbolbaums zu einem seiner Blätter repräsentiert eine Grammatikregel. Das erste Symbol auf diesem Pfad stellt die linke Seite der Regel dar und alle anderen ergeben in der vorgegebenen Reihenfolge eine gültige Folge von Grammatiksymbolen auf der rechten Regelseite.

Die Struktur eines Symbolbaumknotens ist wie folgt definiert:

```
struct knoten_struktur {
      char *zeichenkette;                  /* Grammatiksymbol */
```

Objekt	→ **Konzept**, **Instanz**, **modifiziertes Konzept**
Modalität	→ **Modalitätsbeschreibung**
⋮	→ ...

Abbildung 2.11: Datenstruktur **Eintragsinformation** der ERNEST–Kontrolle

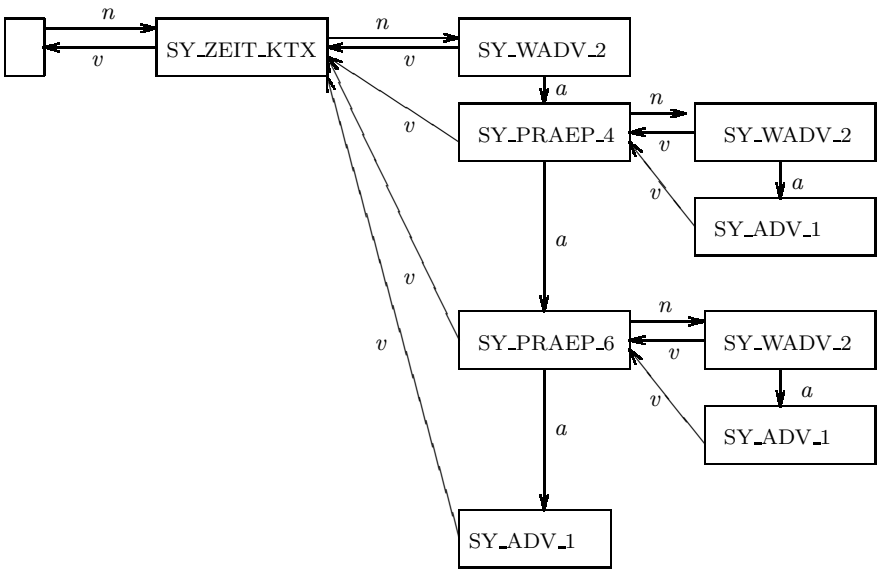


Abbildung 2.12: Symbolbaum zum Zwischenspeichern einer Grammatikregel. Die Kantenbeschriftungen bedeuten:
a: Alternative
v: Vorgänger
n: Nachfolgerliste

```
    struct knoten_struktur *vorgaenger;  /* Verweis auf direkten Vor-
                                            gaenger des Grammatiksymbols */
    struct knoten_struktur *liste;       /* Nachfolgerliste */
    struct knoten_struktur *alternative; /* Verweis auf die naechste
                                            Alternative innerhalb einer
                                            Nachfolgerliste*/
};

typedef struct knoten_struktur baum_knoten;
```

Beispiel: Die zusammengesetzte Grammatikregel

$$
\begin{array}{lcl}
SY\_ZEIT\_KTX\_0 & : & SY\_WADV\_2 \\
 & | & SY\_PRAEP\_4\ SY\_WADV\_2 \\
 & | & SY\_PRAEP\_4\ SY\_ADV\_1 \\
 & | & SY\_PRAEP\_6\ SY\_WADV\_2 \\
 & | & SY\_PRAEP\_6\ SY\_ADV\_1 \\
 & | & SY\_ADV\_1 \\
 & ; &
\end{array}
$$

wird im Symbolbaum der Abbildung 2.12 zwischengespeichert.

Wenn ein Knoteneintrag eines Suchbaumknotens keine Nachfolger besitzt, bedeutet dies, daß das zugrundeliegende Konzept aus der Hypothesenebene stammt. Aus der Eintragsinformation ist abzu-

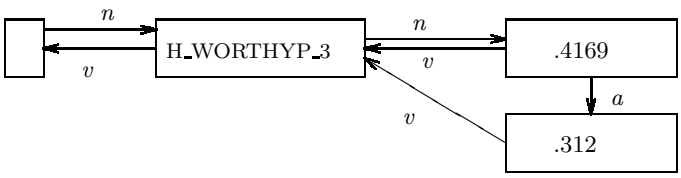


Abbildung 2.13: Symbolbaum zum Zwischenspeichern einer Grammatikregel. Die Kantenbeschriftungen bedeuten:
a: Alternative
v: Vorgänger
n: Nachfolgerliste

lesen, auf welche Wörter der Gültigkeitsbereich diese Konzepts eingeschränkt ist. Die dazugehörigen Wortnummern ergeben dann die Terminalsymbole der rechten Seite der aus diesem Eintrag erzeugten Grammatikregel.

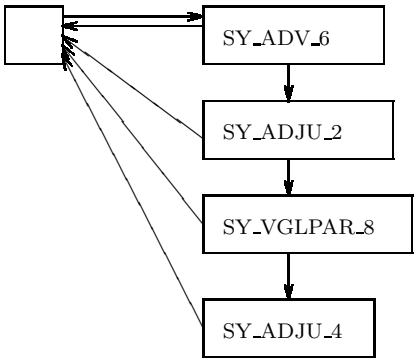
Beispiel: Ist der Gültigkeitsbereich des modifizierten Konzepts H_WORTHYP mit der Identifikation 3 auf die Wörter mit den Nummern 4169 und 312 eingeschränkt, ergibt das folgende Regel:

```
H_WORTHYP_3     : .4169
                | .312
                ;
```

Abbildung 2.13 zeigt den dazugehörigen Symbolbaum.

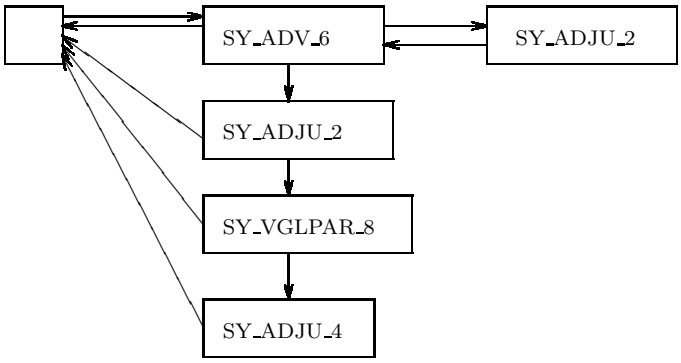
Aus einem Knoteneintrag, dessen Nachfolgerliste nicht leer ist, wird eine zusammengesetzte Regel erzeugt, deren rechte Seite aus Folgen von Nichtterminalsymbolen gebildet wird. Die Bestandteile dieser Folgen werden durch die Nachfolger des Knoteneintrags bestimmt. Diese Nachfolger repräsentieren Bestandteile und Konkretisierungen des Konzepts, das dem ausgewählten Knoteneintrag und der diesem Eintrag zugeordneten Modalitätsbeschreibung zugrundeliegt. Aus den Nachfolgeeinträgen werden Nichtterminalsymbole gewonnen, indem den Namen der entsprechenden modifizierten Konzepte die jeweilige Identifikation hinzugefügt wird. Jeder Nachfolgeeintrag ist mit dem ausgewählten Knoteneintrag durch eine Bestandteils- oder eine Konkretisierungskante verbunden. Mit Hilfe der Rollennamen dieser Kanten und der Modalitätsbeschreibung, die dem ausgewählten Knoteneintrag zugeordnet ist, werden die verschiedenen Folgen von Nichtterminalsymbolen auf der rechten Seite der zu erzeugenden Grammatikregel bestimmt. Es wird dabei ein Teilsymbolbaum wie folgt gebildet: Begonnen wird mit jeder Kante der Modalitätsbeschreibung. Jede dieser Kanten verweist auf einen Nachfolgeeintrag, aus dem wie oben beschrieben ein Nichtterminalsymbol gewonnen wird. Diese Nichtterminalsymbole werden in die Nachfolgerliste des leeren Wurzelknotens des Teilsymbolbaums eingetragen.

Beispiel: Betrachtet wird das modifizierte Zielkonzept des Suchbaumknoten der Abbildung 2.8. Der Aufbau des Teilsymbolbaums beginnt mit:

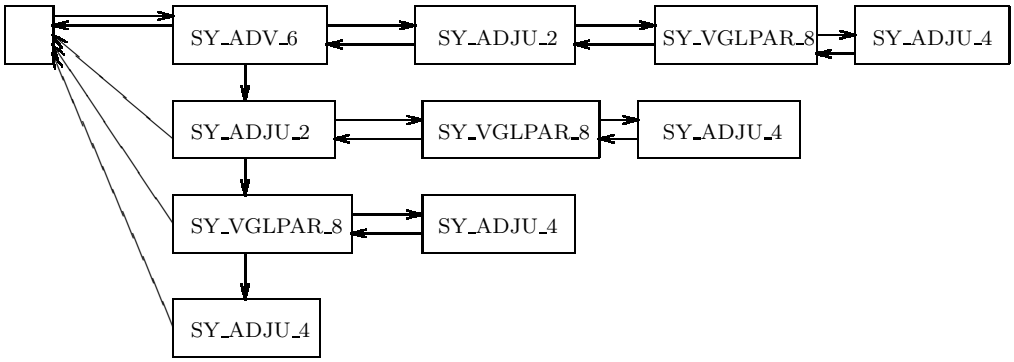


Zu jeder Kante werden dann gemäß der Adjazenzmatrix der Modalitätsbeschreibung alle Nachfolgekanten berechnet. Entsprechend werden zu den bereits erzeugtenSymbolbaumknoten Nachfolgerlisten generiert.

Beispiel: Gemäß der Adjazenzmatrix des modifizierten Konzepts SY_ZEIT_DAU in der Abblidung 2.8 besitzt die Kante *so* die Nachfolgekante *kurz_lang*. Der Teilsymbolbaum wird entsprechend ergänzt:



Nach der vollständigen Bearbeitung der Adjazenzmatrix sieht der Teilsymbolbaum dann so aus:



Die Anzahlschranken für die einzelnen Kanten werden bei der Auswertung der Adjazenzmatrix berücksichtigt. Da bisher nicht beachtet wurde, ob die erzeugten Ketten von Nichtterminalsymbolen alle als obligatorisch gekennzeichneten Bestandteile und Konkretisierungen enthalten oder nicht, muß der Teilsymbolbaum am Ende korrigiert werden. Jede Kette von Nichtterminalsymbolen, die durch einen Pfad im Teilsymbolbaum vom Wurzelknoten zu einem Blattknoten gegeben ist und die nicht alle obligatorischen Symbole enthält, wird entfernt.

Beispiel: Der vollständige Teilsymbolbaum aus dem vorigen Beispiel wird zu folgendem Teilsymbolbaum korrigiert:

Falls einem Knoteneintrag mit Nachfolgern keine Modalitätsbeschreibung zugeordnet ist, ist die Reihenfolge der Nichtterminalsymbole auf der rechten Seite der zu erzeugenden Regel beliebig. Aus den Symbolen, die aus allen Nachfolgeeinträgen bestimmt werden, werden sämtliche Permutationen in den Teilsymbolbaum eingetragen.

Nach Bearbeitung aller Suchbaumblattknoten kann es vorkommen, daß mehrere Grammatikregeln erzeugt worden sind, deren rechte Seiten identisch sind. Zum Beispiel:

```
H_WORTHYP_3     : .4169
                | .312
                ;
H_WORTHYP_5     : .4169
                | .312
                ;
```

In diesem Fall ist eine der Regeln ausreichend, wenn die Nichtterminalsymbole der linken Seiten der anderen Regeln (hier zum Beispiel H_WORTHYP_5) durch das Nichtterminalsymbol der linken Seite einer Regel (hier H_WORTHYP_3) in der gesamten Grammatik beziehungsweise im gesamten Symbolbaum ersetzt werden. Durch den Ersetzungsvorgang können erneut Regeln mit gleicher rechter Seite entstehen. Diese werden genauso bearbeitet.

Beispiel: Die Grammatik, die zum Konzept SY_ZEIT_DAU erzeugt wird, sieht nach einer Zusammenfassung identischer Regeln so aus:

```
%token .286
%token .312
%token .368
%token .370
%token .375
%token .377
%token .378
%token .380
%token .384
%token .3837
%token .3851
%token .4169

%%

SY_ZEIT_DAU_0   : SY_ADV_6 SY_ADJU_2 SY_VGLPAR_8 SY_ADJU_4
                | SY_ADJU_2
                | SY_VGLPAR_8 SY_ADJU_2
                | SY_ADV_11 SY_ADJU_2
                ;
SY_ADJU_2       : H_WORTHYP_3
                ;
H_WORTHYP_3     : .4169
                | .312
                ;
SY_ADJU_4       : H_WORTHYP_5
                ;
H_WORTHYP_5     : .286
                ;
SY_ADV_6        : H_WORTHYP_7
                ;
H_WORTHYP_7     : .370
                ;
SY_VGLPAR_8     : H_WORTHYP_9
                ;
H_WORTHYP_9     : .368
                ;
SY_ADV_11       : H_WORTHYP_14
                ;
H_WORTHYP_14    : .3837
                | .3851
                | .384
                | .375
                | .380
                | .378
                | .377
                | .370
                ;
```

Am Ende des ganzen Algorithmus zur Grammatikextraktion wird die kontexfreie Grammatik entsprechend des erzeugten Symbolbaums im BISON–Format in eine Ausgabedatei geschrieben. Abbildung 2.14 faßt den Ablauf dieses Algorithmus in einem Struktogramm zusammen.

Suchbaum und Ergebnisspeicher einlesen
Symbolbaum initialisieren
FOR alle Blattknoten des Suchbaums
FOR alle Einträge des Suchbaumknotens
IF Eintrag hat Nachfolger oder zugrundeliegendes Konzept ist aus der Hypothesenebene
THEN Grammatikregel generieren und in den Symbolbaum eintragen
mehrere identische Regeln auf eine reduzieren
Grammatikregeln in eine Ausgabedatei schreiben

Abbildung 2.14: Algorithmus zur Extraktion einer kontextfreien Grammatik

Kapitel 3

Automatisches Generieren von Konstituentenhypothesen unter Einsatz der kontextfreien Grammatik

Im vorigen Kapitel wurde ausführlich beschrieben, wie aus dem ERNEST–Netzwerk des Sprachverarbeitungssystems EVAR automatisch eine kontextfreie Grammatik für die Zeitangaben extrahiert wird. Diese Grammatik definiert alle syntaktisch korrekten Wortketten, die Zeitangaben darstellen. Um nun korrekte Wortfolgen aufbauen zu können, muß man zu jeder syntaktisch korrekten Teilkette ermitteln, mit welchen Wörtern die Teilkette fortgesetzt werden kann. Dazu wird hier ein Parser zu Hilfe genommen.

Unter dem Betriebssystem UNIX steht der Compilergenerator BISON, eine Weiterentwicklung des YACC, zur Verfügung. Im Abschnitt 3.1 wird die Funktionsweise des von BISON erzeugten Parsers erklärt und wie dieser modifiziert werden muß, um mit seiner Hilfe korrekte Wortfolgen generieren zu können.

Um im Sprachsignal gezielt nach syntaktisch korrekten Wortfolgen suchen zu können, werden die Wortfolgen von links nach rechts aufgebaut und, ausgehend von einem bestimmten Startpunkt, mit dem Sprachsignal verglichen. Dieses Signal wird in Segmente unterteilt, wobei jedes dieser Segmente als möglicher Anfangspunkt für eine vorgegebene Wortkette dienen kann. Beim Vergleich der erlaubten Wortfolgen mit dem Sprachsignal wird die Ähnlichkeit zwischen beiden bewertet. Gesucht sind schließlich Wortketten, die möglichst gut bewertet sind, also möglichst gut mit dem Sprachsignal übereinstimmen. Aus ihnen werden Wortkettenhypothesen, sogenannte Konstituentenhypothesen, gebildet. Eine solche Hypothese besteht aus einer eindeutigen Identifikation, der Wortkette, dem Anfangs- beziehungsweise Endpunkt im Sprachsignal und einer Bewertung.

Im Abschnitt 3.2 wird erläutert, wie die automatische Generierung von Konstituentenhypothesen im einzelnen abläuft, wobei der modifizierte BISON–Parser eingesetzt wird, um Wortfolgen schrittweise aufbauen zu können.

3.1 Der BISON–Compilergenerator und der BISON–Parser

Üblicherweise findet ein Parser Anwendung in der Syntaxanalyse. Eine Folge von Eingabesymbolen wird auf syntaktische Korrektheit überprüft. Dabei wird anhand einer Grammatik untersucht, ob aus einem Startsymbol unter Anwendung der einzelnen Regeln der zugrundeliegenden Grammatik die gegebene Kette von Eingabesymbolen abgeleitet werden kann. Es gibt zwei verschiedene Vorgehensweisen. Bei der Bottom–up–Analyse werden, ausgehend von der Eingabe, Grammatikregeln umgekehrt angewendet und versucht, die Eingabekette zu reduzieren, bis das Startsymbol erreicht ist. Anders wird bei der Top–down–Analyse das Startsymbol ersetzt, bis die gesuchte Folge abgeleitet ist. Der BISON–Compilergenerator produziert einen $LALR(1)$–Parser, der eine Bottom–up–Analyse durchführt. Die genaue Vorgehensweise der $LALR(1)$–Analyse wird im Abschnitt 3.1.1 beschrieben. Genauere Einzelheiten finden sich in [AhJo74], [Maye86].

In der hier interessierenden Anwendung müssen nun gegebene korrekte Wortfolgen so erweitert werden, daß keine Fehler entstehen. Die erweiterte Wortfolge muß also wieder syntaktisch korrekt

sein. Im Abschnitt 3.1.2 wird erläutert, welche Informationen aus dem Parser dazu nötig sind, und inwiefern der Parser modifiziert werden muß, um syntaktisch korrekte Wortfolgen produzieren zu können. Um einen so geänderten Parser automatisch zu generieren, müssen die nötigen Modifikationen am Compilergenerator selbst vorgenommen werden.

3.1.1 $LR(1)$- und $LALR(1)$–Analyse

Durch eine Grammatik wird die syntaktische Struktur einer Menge von Sätzen festgelegt. Diese Sätze bilden die Sprache, die durch die Grammatik definiert ist. Eine Grammatik ist dabei durch ein Quadrupel $G = (N, T, P, S)$ definiert, wobei

- N eine Menge von Nichtterminalsymbolen,
- T eine Menge von Terminalsymbolen,
- P eine endliche, nichtleere Menge von Produktionen und
- S ein spezielles Nichtterminalsymbol, das Startsymbol sind.

Terminalsymbole sind hierbei diejenigen Symbole, aus denen die Sätze der durch die Grammatik festgelegten Sprache ausschließlich bestehen.

Die Menge der Produktionen P ist eine Teilmenge von $V^* \times V^*$, wobei $V = N \cup T$ die Menge aller Grammatiksymbole ist. Eine Produktion oder Regel der Grammatik wird geschrieben als $x \longrightarrow y$, wobei $x \in V^*$ die linke und $y \in V^*$ die rechte Seite der Regel sind.

Durch Anwendung dieser Regeln kann nun iterativ aus dem Startsymbol die Sprache einer solchen Grammatik erzeugt werden. Dabei werden Folgen aus Grammatiksymbolen produziert, indem Zeichen, die die linke Seite einer Regel bilden, durch die Zeichen auf der rechten Seite dieser Regel ersetzt werden. Man spricht von direkter Ableitbarkeit und Ableitbarkeit, wobei man unter diesen Begriffen folgendes versteht:

- Direkte Ableitbarkeit:
  u ist direkt ableitbar nach v, geschrieben $u \longrightarrow v$,
  wenn $u = x\alpha y$, $v = x\beta y$ und $\alpha \longrightarrow \beta \in P$ mit $u, v, x, y \in V^*$.
- Ableitbarkeit:
  u ist ableitbar nach v, geschrieben $u \longrightarrow^* v$,
  wenn $u = x_0 \longrightarrow x_1 \longrightarrow \ldots \longrightarrow x_n = v$ mit $1 \leq i \leq n$.

Die durch eine Grammatik G erzeugbare Sprache $L(G)$ besteht dann aus allen Symbolfolgen, die aus dem Startsymbol ableitbar sind und ausschließlich aus Terminalsymbolen afgebaut sind, also:

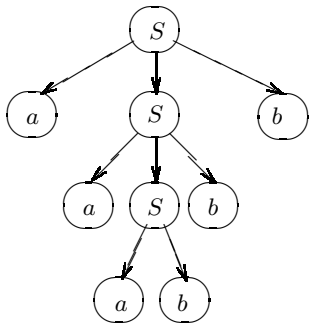
$$L(G) = \{w | w \in T^* \text{und} S \longrightarrow^* w\}$$

Beispiel: $G_1 = (N, T, P, S)$ mit $N = \{S\}$, $T = \{a, b\}$, $P = \{S \longrightarrow aSb, S \longrightarrow ab\}$;
Die durch G_1 erzeugte Sprache ist $L(G_1) = \{a^n b^n | n \geq 1\}$.

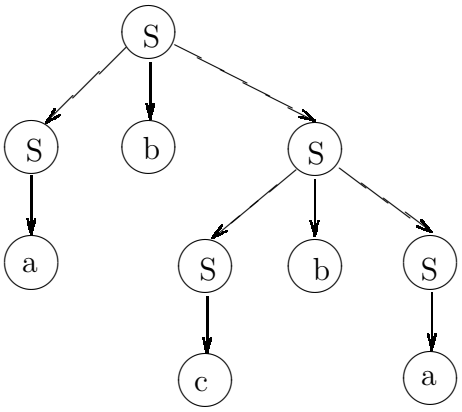
Durch Einschränkungen der in P zugelassenen Produktionen erhält man verschiedene Grammatiktypen:

- allgemeine Grammatiken: keine Einschränkungen;
- kontextsensitive Grammatiken: aus $u \longrightarrow v \in P$ folgt $|u| \leq |v|$;
- kontextfreie Grammatiken: aus $u \longrightarrow v \in P$ folgt $u \in N$ und $v \in V^*$;
- reguläre Grammatiken: aus $u \longrightarrow v \in P$ folgt $u \in N$ und $v \in NT^* \cup T^*$ (linkslineare Grammatik) oder $v \in T^*N \cup T^*$ (rechtslineare Grammatik);

Für die durch die verschiedenen Grammatiktypen erzeugten Sprachen gilt folgender Zusammenhang, wobei die angegebenen Inklusionen echt sind:



(a)



(b)

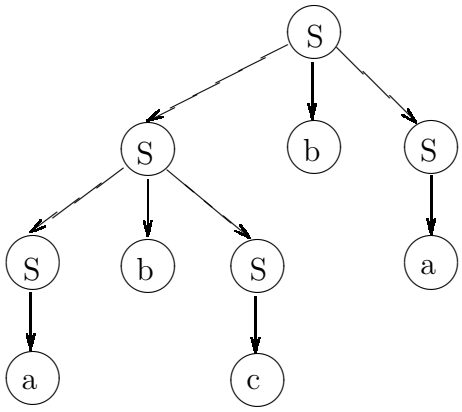


Abbildung 3.1: Zwei Ableitungsbäume für den Satz *abcba* der Sprache $L(G_2)$

allg. Spr. $\supset$ kontexsensitive Spr. $\supset$ kontextfreie Spr. $\supset$ reguläre Spr.

Im weiteren werden hier nur kontextfreie Grammatiken behandelt.

Durch Anwendung der Produktionsregeln wird aus dem Startsymbol S ein Satz aus der Sprache $L(G)$ abgeleitet. Bei diesem Vorgang kann ein sogenannter **Ableitungsbaum** konstruiert werden. Zur Veranschaulichung soll wieder obiges Beispiel betrachtet werden: Der Satz *aaabbb* läßt sich folgendermaßen erzeugen: $S \longrightarrow aSb \longrightarrow aaSbb \longrightarrow aaabbbb \in T^*$. Der dazugehörige Ableitungsbaum sieht dann folgendermaßen aus:

Jeder Knoten eines solchen Baumes repräsentiert ein Grammatiksymbol. Die Blätter, von links nach rechts gelesen, ergeben den abgeleiteten Satz.

Gibt es zu jedem Satz einer, aus einer Grammatik G erzeugten Sprache $L(G)$ genau einen Ableitungsbaum, dann ist G **eindeutig**. Die Grammatik G ist **mehrdeutig**, falls sie Sätze produzieren kann, zu denen es zwei oder mehr Ableitungsbäume gibt.

Beispiel: $G_2 = (N, T, P, S)$ mit $N = \{S\}$, $T = \{a, b, c\}$,
$P = \{S \longrightarrow SbS, S \longrightarrow a, S \longrightarrow c\}$;
Der Satz *abcba* kann auf zwei verschiedene Arten erzeugt werden. Der Unterschied besteht in der Reihenfolge, in der die einzelnen Regeln angewendet werden, da nach der ersten direkten Ableitung $S \longrightarrow SbS$ auf die Symbolfolge SbS zwei Regeln angewendet werden können, ohne beim weiteren Verlauf der Analyse auf einen Fehler zu stoßen. Dem Satz *abcba* entsprechen daher die beiden Ableitungsbäume (a) und (b) aus Abbildung 3.1.

Ein Programm, das einen vorgegebenen Satz daraufhin überprüft, ob er zu einer durch eine vorgegebene Grammatik erzeugten Sprache gehört oder nicht, nennt man einen **Parser**. Es gibt verschiedene

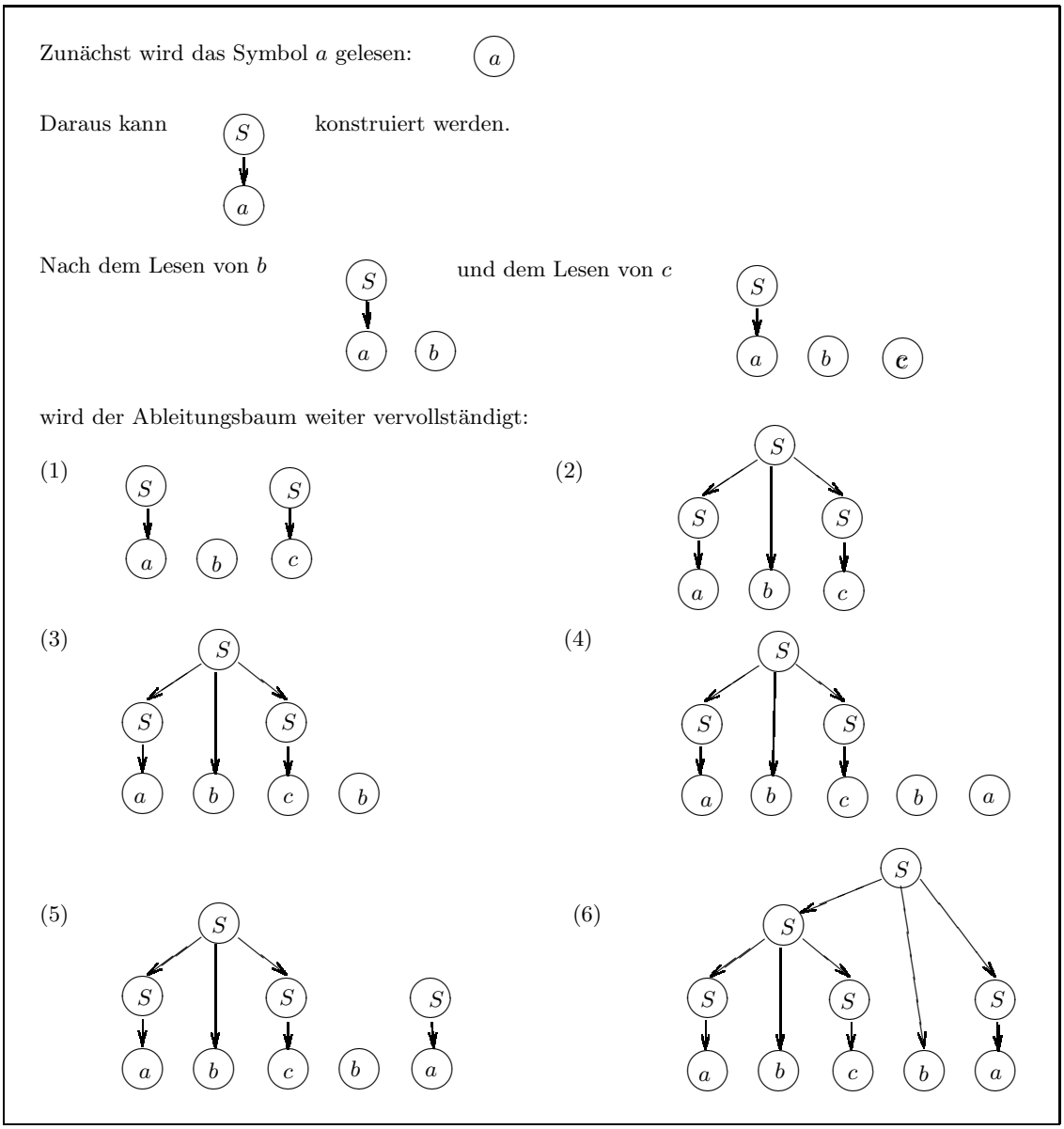


Abbildung 3.2: Analyse des Wortes *abcba* aus der Sprache $L(G_2)$

Möglichkeiten, wie ein Parser arbeitet. Hier sollen nur *LR*–Parser betrachtet werden, wobei das L für "left-to-right scan of the input" (Analyse der Eingabe von links nach rechts) und das R für "rightmost derivation" (Ableitung des am weitesten rechts stehenden Nichtterminalsymbols zuerst) stehen.

Die Vorgehensweise bei der Analyse eines *LR*–Parsers kann man sich folgendermaßen vorstellen: Der zu untersuchende Satz steht auf einem Eingabeband, das Symbol für Symbol von links nach rechts gelesen wird. Dabei versucht der Parser einen Ableitungsbaum von unten her ("bottom-up") aufzubauen.

Beispiel: Die Abbildung 3.2 zeigt, wie bei der *LR*–Analyse der Satz *abcba* der Sprache $L(G_2)$ der dazugehörige Ableitungsbaum mit Hilfe der Regeln der Grammatik G_2 aufgebaut wird.

Auf diese Art und Weise werden in umgekehrter Reihenfolge Rechtsableitungen konstruiert. Rechtsableitungen sind Ableitungen, bei denen immer das am weitesten rechts stehende Nichtterminalsymbol ersetzt wird.

Ein Parser muß also, um einen vollständigen Ableitungsbaum zu konstruieren, zwischen vier möglichen Aktionen unterscheiden:

- **shift:** das nächste Eingabesymbol lesen und einen neuen Knoten, markiert mit diesem Symbol, dem bisher teilweise aufgebauten Ableitungsbaum hinzufügen.
- **reduce:** reduzieren mit einer Regel der Form $A \longrightarrow X_1 X_2 \dots X_n$ mit $A \in N$ und $X_1, X_2, \dots X_n \in V$, das heißt einen neuen Knoten, markiert mit A, kreieren und diesen mit den n am weitesten rechts stehenden Wurzelknoten verbinden.
- **accept:** Analyse erfolgreich beenden, wenn kein Symbol mehr auf dem Eingabeband steht und ein vollständiger Ableitungsbaum konstruiert worden ist.
- **error:** Fehlermeldung.

Welche dieser Aktionen der Parser in einer gegebenen Situation auszuführen hat, hängt davon ab, welche Aktionen bisher ausgeführt wurden und welche Symbole als nächstes auf dem Eingabeband stehen. Wenn nur das nächste Symbol auf dem Eingabeband betrachtet wird, um eine Entscheidung zu treffen, ist der Parser ein $LR(1)$–Parser. Werden hingegen die nächsten k ($k \geq 0$) Symbole betrachtet, handelt es sich um einen $LR(k)$–Parser.

Die bisher ausgeführten Analyseaktionen und das nächste Eingabesymbol, das sogenannte Lookahead-Symbol, bestimmen den **Zustand**, in dem sich ein $LR(1)$–Parser in einer bestimmten Situation befindet. Die möglichen Zustände und die Aktionen, die ein Parser in den einzelnen Zuständen bei Vorliegen der verschiedenen möglichen Lookahead-Symbole ausführen soll, können berechnet und in Tabellen, den sogenannten Analysetabellen, zusammengefaßt werden. Im folgenden soll gezeigt werden, wie dies geschieht:

Eine **Rechtssatzform** ist eine Folge von Grammatiksymbolen, die ausschließlich durch Anwendung von Rechtsableitungen aus dem Startsymbol entstanden ist, das heißt, es wurde, beginnend mit dem Startsymbol, immer das am weitesten rechts stehende Nichtterminalsymbol ersetzt. Ein **lebensfähiges Präfix** ist eine Zeichenfolge, die zu einer Rechtssatzform ergänzt werden kann.

Beispiel: *ab* ist ein lebensfähiges Präfix der Grammatik G_2, da *ab* zur Rechtssatzform *abcba* vervollständigt werden kann.

Die Zustände eines LR–Parsers werden nun als Klassen solcher lebensfähiger Präfixe aufgefaßt. Zu jedem Zeitpunkt der Analyse muß die Zeichenfolge, die durch Aneinanderreihung der Symbole der bisher kreierten Wurzelknoten entsteht, ein lebensfähiges Präfix sein.

Eine **Auskunft** ist, einfach gesagt, eine Produktion, deren rechte Seite durch einen Punkt unterteilt ist und die in eckige Klammern eingeschlossen ist. Die Auskunft $[A \longrightarrow \alpha.\beta]$ hat dabei folgende Bedeutung: Eine Eingabezeichenfolge, die aus α ableitbar ist, wurde bereits betrachtet. Sie entspricht den am weitesten rechts stehenden Wurzelknoten des bis dahin erzeugten Ableitungsbaums. Es kann mit der Produktion $A \longrightarrow \alpha\beta$ reduziert werden, falls als nächstes eine Zeichenfolge kommt, die aus β ableitbar ist. Wenn der Anfang der Eingabe zu dem lebensfähigen Präfix $\gamma\alpha$ reduziert worden ist und auch γA ein lebensfähiges Präfix bildet, dann ist die Auskunft $[A \longrightarrow \alpha.\beta]$ eine **gültige Auskunft** für $\gamma\alpha$. Die Menge aller Auskünfte, die in einer bestimmten Phase der Analyse gültig sind, entsprechen einem Zustand des Parsers. Jede mögliche solche Menge ergibt einen Zustand und da es nur endlich viele Produktionen, also auch nur endlich viele Auskünfte gibt, können nur endlich viele Zustände aus Auskunftsmengen gebildet werden. Die Menge der Auskünfte $V(\gamma)$, die für das Präfix γ gültig sind, nennt man **Information** zu γ.

Um die Zustände eines $LR(0)$–Parsers zu erzeugen, wird zunächst die zugrundeliegende Grammatik um die Regel $ACCEPT \longrightarrow S$ erweitert. Eine Reduktion mit dieser Regel bedeutet, daß die Analyse erfolgreich beendet ist. Als nächstes wird die Menge $V(\epsilon)$ berechnet, wobei ϵ für das leere Wort steht. Aus dieser Menge werden alle weiteren Informationen mit nachfolgender Vorschrift bestimmt. Die Menge $V(\epsilon)$ enthält am Anfang die Auskunft $[ACCEPT \longrightarrow .S]$, da auf dem verbleibenden Eingabeband ein Satz erwartet wird, der aus S ableitbar ist. Mit dem zweiten Teil der folgenden Vorschrift wird die Menge $V(\epsilon)$ vervollständigt. Wenn nun $V(\gamma)$ berechnet worden ist, laßt sich daraus $V(\gamma X)$ mit $X \in N \cup T$ berechnen:

1. Für jede Auskunft der Form $[A \longrightarrow \alpha.X\beta]$ in $V(\gamma)$ wird eine Auskunft der Form $[A \longrightarrow \alpha X.\beta]$ in $V(\gamma X)$ aufgenommen.

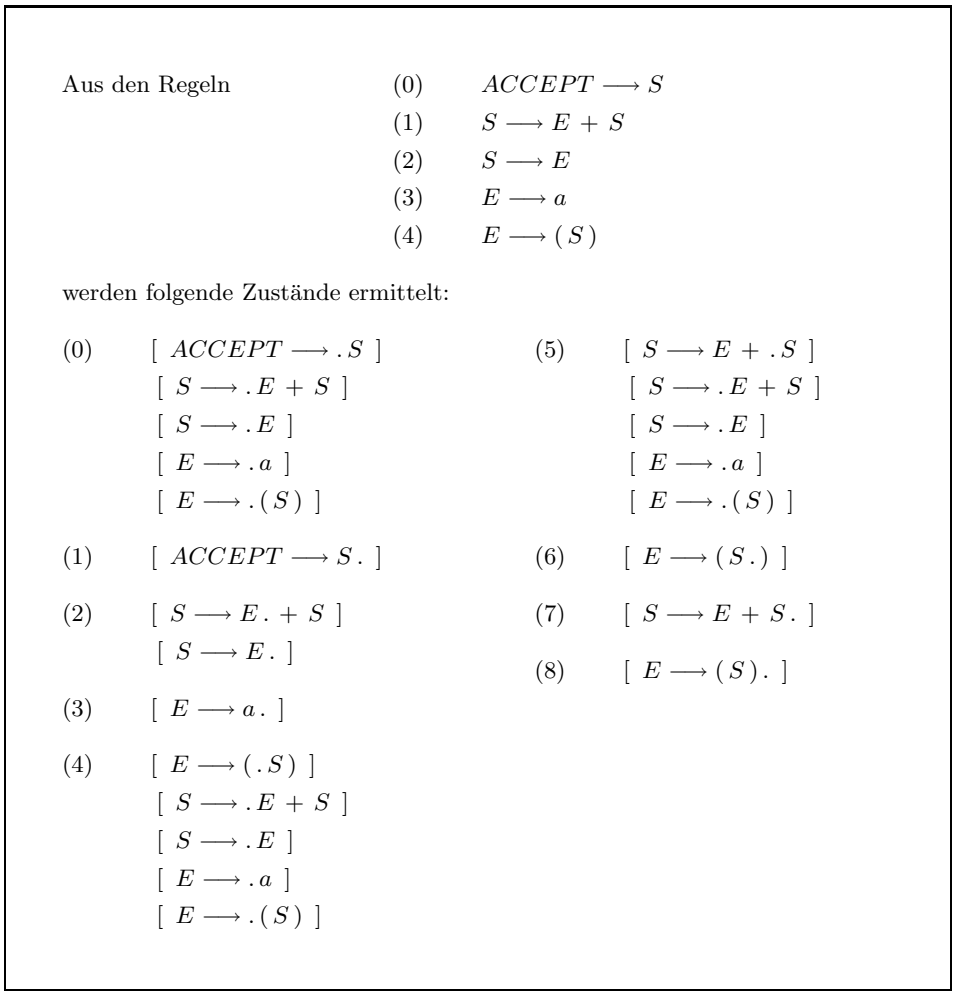


Abbildung 3.3: Zustände eines $LR(0)$–Parsers zur Grammatik G_3

2. Für jede Auskunft der Form $[B \longrightarrow \alpha.C\beta]$ in $V(\gamma X)$, wobei C ein Nichtterminalsymbol ist, werden in $V(\gamma X)$ die Auskünfte $[C \longrightarrow .\alpha_1]$, $[C \longrightarrow .\alpha_2]$, $\ldots [C \longrightarrow .\alpha_n]$ hinzugenommen. $C \longrightarrow \alpha_1$, $C \longrightarrow \alpha_2$, $\ldots C \longrightarrow \alpha_n$ sind alle Produktionen mit C als linker Seite. Dieser Vorgang wird solange wiederholt, bis keine neuen Auskünfte mehr gefunden werden.

Man kann zeigen, daß mit dieser Vorschrift genau diejenigen Auskünfte berechnet werden, die für γX gültig sind (siehe [AhUl72]). Die Zustände werden nun mit den unterschiedlichen Mengen $V(\gamma)$ gleichgesetzt. Zwei oder mehr identische Mengen faßt man zu einem Zustand zusammen.

Beispiel: In Abbildung 3.3 sind die $LR(0)$–Zustände mit den entsprechenden Auskünften für die Grammatik $G_3 = (N, T, P, S)$ aufgeführt, mit $N = \{S, E\}$, $T = \{a, +, (,)\}$ und $P = \{S \longrightarrow E + S,\ S \longrightarrow E,\ E \longrightarrow a,\ E \longrightarrow (S)\}$.

Aus dieser Konstruktion von Parserzuständen läßt sich leicht eine sogenannte *goto*-Funktion ableiten, die in einer Tabelle dargestellt werden kann. Die *goto*-Funktion wird iterativ definiert, wobei Z die Menge aller Zustände, $z \in Z$, $\gamma \in V^*$ und $X \in N \cup T$ ist:

$$\begin{array}{lcl} goto & : & Z \times V^* \longrightarrow Z \\ \\ goto\,(z, \epsilon) & = & z \\ goto\,(z, \gamma X) & = & goto\,(\,goto(z, \gamma)\,, X) \end{array}$$

Dabei gilt: Wenn $z = V(\alpha)$, dann $z = goto(z_0, \alpha)$ mit $z_0 = V(\epsilon)$.

Goto-Tabelle

	S	E	a	$+$	(	)
0	1	2	3	—	4	—
1	—	—	—	—	—	—
2	—	—	—	5	—	—
3	—	—	—	—	—	—
4	6	2	3	—	4	—
5	7	2	3	—	4	—
6	—	—	—	—	—	8
7	—	—	—	—	—	—
8	—	—	—	—	—	—

Analyseaktions-Tabelle

	a	$+$	(	)	\$
0	shift	error	shift	error	error
1	error	error	error	error	accept
2	reduce 2	shift / reduce 2	reduce 2	reduce 2	reduce 2
3	reduce 3	reduce 3	reduce 3	reduce 3	reduce 3
4	shift	error	shift	error	error
5	shift	error	shift	error	error
6	error	error	error	shift	error
7	reduce 1	reduce 1	reduce 1	reduce 1	reduce 1
8	reduce 4	reduce 4	reduce 4	reduce 4	reduce 4

Abbildung 3.4: Goto- und Analyseaktionstabellen eines $LR(0)$–Parsers zur Grammatik G_3

Daraus läßt sich dann die **Goto-Tabelle** konstruieren, in der zu jedem möglichen Zustand z und zu jedem möglichen Grammatiksymbol $sym \in N \cup T$ der Wert $goto(z, sym) = goto(z_0, \alpha\, sym)$ für alle $\alpha \in V^*$ angegeben wird.

Für die Analyse ist es aber nicht nur von Bedeutung, in welchen Zustand der Parser nach Ausführung einer Aktion übergehen muß, sondern auch, welche Aktion er in einem gegebenen Zustand ausführen soll. Dazu wird eine **Analyseaktionstabelle** aus den berechneten Informationen, also den Zuständen ermittelt. In dieser Tabelle ist für jeden Zustand und für jedes Symbol, das als nächstes auf dem Eingabeband erscheint, die Aktion eingetragen, die dann ausgeführt werden soll. Folgende Vorschrift kommt dabei zur Anwendung:

- Wenn ein Zustand z eine Auskunft der Form $[A \longrightarrow \alpha.a\beta]$ mit $a \in T$ enthält, heißt das, daß der Parser in diesem Zustand die Operation *shift* ausführen soll, wenn auf dem Eingabeband als nächstes Zeichen ein a steht. Außerdem soll er in den Zustand $goto(z, a)$ übergehen.
- Eine Auskunft der Form $[A \longrightarrow \alpha.]$ bedeutet, daß mit der Regel $A \longrightarrow \alpha$ reduziert werden soll (*reduce*), wobei es bei einem $LR(0)$–Parser keine Rolle spielt, welches Zeichen als nächstes auf dem Eingabeband erscheint.
- Die Aktion *accept* wird dann ausgeführt , wenn der Parser sich in dem Zustand befindet, der die Auskunft $[ACCEPT \longrightarrow S.]$ enthält, und auf dem Eingabeband nur noch das Zeichen \$ steht, das das Ende der Eingabe markiert.

Beispiel: In Abbildung 3.4 sind die für die Grammatik G_3 berechneten Goto- und Analyseaktionstabellen eines $LR(0)$–Parsers dargestellt.

Ein $LR(0)$–Parser arbeitet nun folgendermaßen: Am Anfang wird der erste Zustand $z_0 = V(\epsilon)$ in einen leeren Keller geschrieben. Der aktuelle Zustand ist jeweils der oberste Zustand auf diesem Keller, also zu Beginn z_0. Während der Analyse führt der Parser jeweils diejenige Analyseaktion (*shift*, *reduce*, *accept*, *error*) aus, die entsprechend dem aktuellen Zustand z und dem nächsten Eingabesymbol sym durch die Analyseaktions-Tabelle gegeben ist. Bei einer *shift*-Aktion wird das Eingabesymbol vom Eingabeband entfernt. Der Zustand, der gemäß der Goto-Tabelle durch den aktuellen Zustand z und das Eingabesymbol sym gegeben ist, wird als neuer aktueller Zustand in den Keller geschrieben. Bei einer *reduce*-Aktion bleibt das Eingabesymbol sym unberührt auf dem Band stehen. In der Analyseaktionstabelle bedeutet nun *reduce* x, daß mit der Regel x reduziert werden muß. Wenn die Regel x von der Form $A \longrightarrow X_1 \ldots X_n$ mit $A \in N$, $X_1 \ldots X_n \in N \cup T$ ist, dann werden im Keller die obersten n Zustände entfernt. Der Zustand, der dann oben auf dem Keller steht und das Symbol A bestimmen mittels der Goto-Tabelle den neuen aktuellen Zustand, der auf den Keller geschrieben

wird. Bei einer *accept*-Aktion wird die Analyse beendet. Bei einer *error*-Aktion schließlich kann die Analyse nicht korrekt fortgesetzt werden und wird daher mit einer Fehlermeldung abgebrochen.

Mit diesem Vorgehen wird also nicht mehr explizit ein Ableitungsbaum konstruiert, sondern nur noch die Zustände über einen Keller verwaltet. Die zugrundeliegende Idee bleibt jedoch erhalten.

Die Konstruktion eines $LR(0)$–Parsers kann zu verschiedenen Konflikten führen:

- Ein *shift / reduce*–Konflikt tritt dann auf, wenn der Parser in einem bestimmten Zustand bei einem oder mehreren nächstmöglichen Eingabesymbolen nicht entscheiden kann, ob mit einer Grammatikregel reduziert oder das Eingabesymbol geshiftet werden soll.

- Ein *reduce / reduce*–Konflikt liegt dann vor, wenn in einem Zustand mehrere *reduce*–Aktionen möglich sind.

Beispiel: Die Analyseaktionstabelle des $LR(0)$–Parsers für die Grammatik G_3 enthält einen *shift / reduce*–Konflikt (siehe Abbildung 3.4).

Sind die Steuertabellen eines Parsers konfliktfrei, bedeutet das , daß zu jedem Zeitpunkt der Analyse eindeutig festgelegt ist, welche Aktion als nächstes ausgeführt werden soll. Der Parser arbeitet in diesem Fall deterministisch.

Konflikte können gelöst werden, wenn man zur Berechnung der Zustände und der Steuertabellen das nächstmögliche Symbol auf dem Eingabeband mit einbezieht. Dieses Symbol ist das **Lookahead-Symbol**. Man erhält durch diese Vorgehensweise einen $LR(1)$–Parser.

Lookahead-Symbole sind folgendermaßen definiert: Die Auskunft $[A \longrightarrow \alpha \,.\, \beta]$ sei gültig für ein lebensfähiges Präfix $\gamma\,\alpha$. Ein Eingabesymbol a ist dann **erlaubt** für $[A \longrightarrow \alpha \,.\, \beta]$, wenn für ein $w \in T^*$ die beiden Zeichenketten $\gamma\alpha\beta aw$ und γAaw durch Rechtsableitungen aus dem Startsymbol erzeugt werden können. Die Menge von Symbolen, die für jede Auskunft erlaubt sind, bilden die **Lookahead-Menge** für diese Auskünfte.

Die Auskünfte werden nun genau um diese Mengen erweitert. $([A \longrightarrow \alpha \,.\, \beta\,], \{b, c\})$ ist beispielsweise eine Auskunft mit dem Kern $[A \longrightarrow \alpha \,.\, \beta\,]$ und der Lookahead-Menge $\{b, c\}$. Die Berechnung der Zustände eines $LR(1)$–Parsers beginnen mit:

$$([ACCEPT \longrightarrow \,.\, S], \{\$\})$$

Die Zustände eines $LR(0)$–Parsers wurden jeweils in zwei Schriten ermittelt. Diese Schritte werden nun durch die Berechnung der Lookahead-Mengen ergänzt.

- Bei Ausführung des ersten Schrittes wird aus einer Auskunft der Form $([A \longrightarrow \alpha \,.\, X\beta], L)$ die Auskunft $([A \longrightarrow \alpha X \,.\, \beta], L)$ ermittelt.

- Bei Ausführung des zweiten Schritts erhält man aus einer Auskunft der Form $([A \longrightarrow \alpha \,.\, B\beta], L)$ die Auskünfte $([B \longrightarrow \,.\, \delta], L')$, wobei $B \longrightarrow \delta$ eine Grammatikregel ist. L' enthält dabei alle Terminalsymbole, die das erste Symbol eines Satzes sind, der aus βa mit $a \in L$ ableitbar ist.

Innerhalb eines Zustandes werden Auskünfte mit dem gleichen Kern zusammengefaßt. Aus $([A \longrightarrow \alpha \,.\, \beta], L_1)$ und $([A \longrightarrow \alpha \,.\, \beta], L_2)$ wird demnach $([A \longrightarrow \alpha \,.\, \beta], L_1 \cup L_2)$.

Aus den Informationen der $LR(1)$–Analyse wird die entsprechende Aktionstabelle nun folgendermaßen gewonnen:

- Wenn ein Zustand z eine Auskunft der Form $([A \longrightarrow \alpha.a\beta], L)$ mit $a \in T$ enthält, heißt das, daß der Parser in diesem Zustand die Operation *shift* ausführen soll, wenn auf dem Eingabeband als nächstes Zeichen ein a steht.

- Eine Auskunft der Form $([A \longrightarrow \alpha.], L)$ bedeutet, daß mit der Regel $A \longrightarrow \alpha$ reduziert werden soll (*reduce*), wenn auf dem Eingabeband als nächstes ein Zeichen $b \in L$ erscheint.

- Die Aktion *accept* wird dann ausgeführt , wenn der Parser sich in dem Zustand befindet, der die Auskunft $[ACCEPT \longrightarrow S.]$ enthält, und auf dem Eingabeband nur noch das Zeichen \$ steht, das das Ende der Eingabe markiert.

Aus den Regeln

(0) $ACCEPT \longrightarrow S$
(1) $S \longrightarrow E + S$
(2) $S \longrightarrow E$
(3) $E \longrightarrow a$
(4) $E \longrightarrow (S)$

werden folgende Zustände ermittelt:

(0) $([\ ACCEPT \longrightarrow .\, S\ ],\ \{\$\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{\$\})$
$([\ S \longrightarrow .\, E\ ],\ \{\$\})$
$([\ E \longrightarrow .\, a\ ],\ \{+, \$\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+, \$\})$

(1) $([\ ACCEPT \longrightarrow S .\ ],\ \{\$\})$

(2) $([\ S \longrightarrow E .\, + S\ ],\ \{\$\})$
$([\ S \longrightarrow E .\ ],\ \{\$\})$

(3) $([\ E \longrightarrow a .\ ],\ \{+, \$\})$

(4) $([\ E \longrightarrow (\, .S\ )\ ],\ \{+, \$\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{)\})$
$([\ S \longrightarrow .\, E\ ],\ \{)\})$
$([\ E \longrightarrow .\, a\ ],\ \{+,)\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+,)\})$

(5) $([\ S \longrightarrow E + .\, S\ ],\ \{\$\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{\$\})$
$([\ S \longrightarrow .\, E\ ],\ \{\$\})$
$([\ E \longrightarrow .\, a\ ],\ \{+, \$\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+, \$\})$

(6) $([\ E \longrightarrow (S .)\ ],\ \{+, \$\})$

(7) $([\ S \longrightarrow E .\, + S\ ],\ \{)\})$
$([\ S \longrightarrow E .\ ],\ \{)\})$

(8) $([\ E \longrightarrow a .\ ],\ \{+,)\})$

(9) $([\ E \longrightarrow (\, .S)\ ],\ \{+,)\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{)\})$
$([\ S \longrightarrow .\, E\ ],\ \{)\})$
$([\ E \longrightarrow .\, a\ ],\ \{+,)\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+,)\})$

(10) $([\ S \longrightarrow E + S .\ ],\ \{\$\})$

(11) $([\ E \longrightarrow (S) .\ ],\ \{+, \$\})$

(12) $([\ S \longrightarrow E + .\, S\ ],\ \{)\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{)\})$
$([\ S \longrightarrow .\, E\ ],\ \{)\})$
$([\ E \longrightarrow .\, a\ ],\ \{+,)\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+,)\})$

(13) $([\ E \longrightarrow (S .)\ ],\ \{+,)\})$

(14) $([\ S \longrightarrow E + S .\ ],\ \{)\})$

(15) $([\ E \longrightarrow (S) .\ ],\ \{+,)\})$

Abbildung 3.5: Zustände eines $LR(1)$–Parsers zur Grammatik G_3

Beispiel: In den Abbildungen 3.5 und 3.6 sind die Zustände und Steuertabellen eines $LR(1)$–Parsers zur Grammatik G_3 gezeigt.

Grammatiken, zu denen sich ein deterministischer $LR(1)$–Parser konstruieren läßt, nennt man $LR(1)$–Grammatiken. Die aus $LR(1)$–Grammatiken gebildeten Parser enthalten also keine *shift / reduce*– und keine *reduce / reduce*–Konflikte. Die von ihnen erzeugbaren Sprachen heißen $LR(1)$–Sprachen. Für jede kontextfreie Sprache gibt es eine $LR(1)$–Grammatik, welche die Sprache erzeugt (siehe [Maye86]). Es gilt also:

$$LR(0)\text{–Sprachen} \subset LR(1)\text{–Sprachen} = \text{kontextfreie Sprachen}$$

Eine weitere Einschränkung bilden die $LALR$–Sprachen, die eindeutig von $LALR$–Parsern analysiert werden können. Die Konstruktion eines $LALR(1)$–Parsers geht von den Zustänen eines $LR(1)$–Parsers aus. Jedoch werden alle Zustände des $LR(1)$–Parsers zusammengefaßt, deren zugehörige

Goto-Tabelle

	S	E	a	$+$	(	)
0	1	2	3	—	4	—
1	—	—	—	—	—	—
2	—	—	—	5	—	—
3	—	—	—	—	—	—
4	6	7	8	—	9	—
5	10	2	3	—	4	—
6	—	—	—	—	—	11
7	—	—	—	12	—	—
8	—	—	—	—	—	—
9	13	7	8	—	9	—
10	—	—	—	—	—	—
11	—	—	—	—	—	—
12	14	7	8	—	9	—
13	—	—	—	—	—	15
14	—	—	—	—	—	—
15	—	—	—	—	—	—

Analyseaktions-Tabelle

	a	$+$	(	)	\$
0	shift	error	shift	error	error
1	error	error	error	error	accept
2	error	shift	error	error	reduce 2
3	error	reduce 3	error	error	reduce 3
4	shift	error	shift	error	error
5	shift	error	shift	error	error
6	error	error	error	shift	error
7	error	shift	error	reduce 2	error
8	error	reduce 3	error	reduce 3	error
9	shift	error	shift	error	error
10	error	error	error	error	reduce 1
11	error	reduce 4	error	error	reduce 4
12	shift	error	shift	error	error
13	error	error	error	shift	error
14	error	error	error	reduce 1	error
15	error	reduce 4	error	reduce 4	error

Abbildung 3.6: Goto- und Analyseaktionstabelle eines $LR(1)$–Parsers zur Grammatik G_3

Auskünfte sich nur durch die Lookahead–Mengen voneinander unterscheiden. Zwei Zustände mit den Auskünften $(kern, lookahead_1)$ und $(kern, lookahead_2)$ verschmelzen dann zu einem Zustand ($kern$, $lookahead_1 \cup lookahead_2$). Über die Lookahead–Mengen wird also die Vereinigung gebildet. Auf diese Weise werden die Tabellen des $LR(1)$–Parsers erheblich verkleinert, da die Anzahl der Zustände verringert wird. Nicht aus jedem $LR(1)$–Parser aber kann ein $LALR(1)$–Parser konstruiert werden, da durch das Vereinigen von Zuständen neue Konflikte auftreten können. Das heißt, nicht jede $LR(1)$–Sprache ist gleichzeitig $LALR(1)$–Sprache. Es gilt:

$$LR(0)\text{–Sprachen} \subset LALR(1)\text{–Sprachen} \subset LR(1)\text{–Sprachen}$$

Als Beispiel zeigen die Abbildungen 3.7 und 3.8 zur Grammatik G_3 die Zustände und die Analysetabellen des dazugehörigen $LALR(1)$–Parsers. Durch das Zusammenfassen werden hier für die $LALR(1)$–Analyse genauso wenig Zustände berechnet, wie für die $LR(0)$–Analyse. Die Tabellen sind also kleiner als die des $LR(1)$–Parsers in Abbildung 3.6. Der *shift / reduce*–Konflikt des $LR(0)$–Parsers taucht aber nicht mehr auf. Die durch G_3 definierte Sprache ist $LALR(1)$–Sprache, aber nicht $LR(0)$–Sprache.

3.1.2 Modifikationen am BISON–Parser

Der BISON–Compilergenerator erwartet als Eingabe eine kontextfreie Grammatik und erzeugt daraus einen $LALR(1)$–Parser (siehe [John74]). Das erforderliche Format der Eingabegrammatik ist in Kapitel 2.3.1 beschrieben.

Wenn die Grammatik nicht $LALR(1)$ ist, entstehen, wie in Kapitel 3.1.1 beschrieben, bei der Berechnung der Analysetabellen *shift / reduce*– oder *reduce / reduce*–Konflikte. Um dennoch eine deterministische Arbeitsweise des Parsers zu ermöglichen, löst der BISON–Compilergenerator die Konflikte folgendermaßen auf, wodurch allerdings die Menge der Sätze eingeschränkt wird, die von einem so erzeugten Parser erkannt werden:

- Beim Auftreten eines *shift / reduce*–Konflikts wird die *shift*–Aktion vorgezogen.
- Beim Auftreten eines *reduce / reduce*–Konflikts wird mit der Grammatikregel reduziert, die in der Eingabe früher definiert worden ist.

Aus den Regeln

(0) $ACCEPT \longrightarrow S$
(1) $S \longrightarrow E + S$
(2) $S \longrightarrow E$
(3) $E \longrightarrow a$
(4) $E \longrightarrow (S)$

werden folgende Zustände ermittelt:

(0) $([\ ACCEPT \longrightarrow .\, S\ ],\ \{\$\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{\$\})$
$([\ S \longrightarrow .\, E\ ],\ \{\$\})$
$([\ E \longrightarrow .\, a\ ],\ \{+, \$\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+, \$\})$

(1) $([\ ACCEPT \longrightarrow S .\ ],\ \{\$\})$

(2) $([\ S \longrightarrow E .\ + S\ ],\ \{), \$\})$
$([\ S \longrightarrow E .\ ],\ \{), \$\})$

(3) $([\ E \longrightarrow a .\ ],\ \{+,), \$\})$

(4) $([\ E \longrightarrow (\, .S)\ ],\ \{+,), \$\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{)\})$
$([\ S \longrightarrow .\, E\ ],\ \{)\})$
$([\ E \longrightarrow .\, a\ ],\ \{+,)\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+,)\})$

(5) $([\ S \longrightarrow E + .\, S\ ],\ \{), \$\})$
$([\ S \longrightarrow .\, E + S\ ],\ \{), \$\})$
$([\ S \longrightarrow .\, E\ ],\ \{), \$\})$
$([\ E \longrightarrow .\, a\ ],\ \{+,), \$\})$
$([\ E \longrightarrow .\, (S)\ ],\ \{+,), \$\})$

(6) $([\ E \longrightarrow (S .)\ ],\ \{+,), \$\})$

(7) $([\ S \longrightarrow E + S .\ ],\ \{), \$\})$

(8) $([\ E \longrightarrow (S) .\ ],\ \{+,), \$\})$

Abbildung 3.7: Zustände eines $LALR(1)$-Parsers zur Grammatik G_3

Goto-Tabelle

	S	E	a	$+$	$($	$)$
0	1	2	3	—	4	—
1	—	—	—	—	—	—
2	—	—	—	5	—	—
3	—	—	—	—	—	—
4	6	2	3	—	4	—
5	7	2	3	—	4	—
6	—	—	—	—	—	8
7	—	—	—	—	—	—
8	—	—	—	—	—	—

Analyseaktions-Tabelle

	a	$+$	$($	$)$	\$
0	shift	error	shift	error	error
1	error	error	error	error	accept
2	error	shift	error	reduce 2	reduce 2
3	error	reduce 3	error	reduce 3	reduce 3
4	shift	error	shift	error	error
5	shift	error	shift	error	error
6	error	error	error	shift	error
7	error	error	error	reduce 1	reduce 1
8	error	reduce 4	error	reduce 4	reduce 4

Abbildung 3.8: Goto- und Analyseaktionstabelle eines $LALR(1)$-Parsers zur Grammatik G_3

Mit diesen Vorschriften wird ein deterministischer Parser erzeugt, der nicht mehr alle Sätze der Sprache erkennt, die durch die zugrundeliegende Grammatik definiert ist.

Beispiel: Aus dem Konzept *SY_ZEIT_DAU* des ERNEST–Netzes wird folgende Grammatik extrahiert:

```
%token .286
%token .312
%token .368
%token .370
%token .375
%token .377
%token .378
%token .380
%token .384
%token .3837
%token .3851
%token .4169

%%

SY_ZEIT_DAU     : SY_ADV_6 SY_ADJU_2 SY_VGLPAR_8 SY_ADJU_4
                | SY_ADJU_2
                | SY_VGLPAR_8 SY_ADJU_2
                | SY_ADV_11 SY_ADJU_2
                ;
SY_ADJU_2       : H_WORTHYP_3
                ;
H_WORTHYP_3     : .4169
                | .312
                ;
SY_ADJU_4       : H_WORTHYP_5
                ;
H_WORTHYP_5     : .286
                ;
SY_ADV_6        : H_WORTHYP_7
                ;
H_WORTHYP_7     : .370
                ;
SY_VGLPAR_8     : H_WORTHYP_9
                ;
H_WORTHYP_9     : .368
                ;
SY_ADV_11       : H_WORTHYP_14
                ;
H_WORTHYP_14    : .3837
                | .3851
                | .384
                | .375
                | .380
                | .378
                | .377
                | .370
                ;
```

Diese Grammatik definiert die aufgeführten Wortfolgen. Hinter den einzelnen Wörtern sind jeweils die entsprechenden Wortnummern in Klammern angegeben.

> so kurz wie möglich (370 4169 368 286), so lang wie möglich (370 312 368 286), kurz(4169), lang(312), wie kurz (368 4169), wie lang (368 312), besonders kurz (3837 4169), besonders lang (3837 312), erst kurz (3851 4169), erst lang (3851 312), möglichst kurz (384 4169), möglichst lang (384 312), noch kurz (375 4169), noch lang (375 312), nur kurz (380 4169), nur lang (380 312), schon kurz (378 4169), schon lang (378 312), sehr kurz (377 4169), sehr lang (377 312), so kurz (370 4169), so lang (370 312).

Die für die $LALR(1)$–Analyse benötigten Steuertabellen enthalten zwei *reduce / reduce*–Konflikte. Der BISON–Compilergenerator löst bei der Berechnung dieser Tabellen die auftretenden Konflikte wie oben beschrieben auf. Die Wortfolgen

> so kurz (370 4169), so lang (370 312).

kann der auf diese Art erzeugte Parser aber nicht mehr erkennen.

Der Parser, der automatisch aus der Eingabegrammatik erzeugt wird, ist durch die Prozedur *yyparse*() definiert, die eine Folge von Terminalsymbolen auf syntaktische Korrektheit und Vollständigkeit überprüft, also die syntaktische Analyse der Eingabe vornimmt. Als Ergebnis produziert *yyparse*() einen Fehler, falls die Eingabekette nicht syntaktisch korrekt ist. Es wird dabei unterschieden, ob die Kette korrekt, aber nicht vollständig, oder falsch ist. Im anderen Fall wird als Ergebnis die Eingabe akzeptiert. *yyparse*() ruft im Lauf der Bearbeitung die Funktion *yylex*() auf. Mit Hilfe dieser Funktion wird die Eingabefolge zeichenweise gelesen, wobei *yylex*() das jeweils nächste Terminalsymbol auf dem Eingabeband liefert. *yylex*() führt die lexikalische Analyse der Eingabefolge aus.

Während der Generierung von Konstituentenhypothesen wird eine bereits behandelte Wortkette, anfangs die leere Wortkette, um ein Wort erweitert, so daß die neu entstehende Wortkette den Anfang einer korrekten Wortfolge bildet. Es ist also eine Funktion erforderlich, die als Eingabe eine syntaktisch korrekte Wortfolge erhält und als Ergebnis die Menge aller Wörter zurückliefert, mit denen die Eingabefolge gemäß einer Grammatik fortgesetzt werden kann. Sie ist folgendermaßen definiert:

```
yyparse_mod(wnr_liste,anzahl,erg_anzahl,erg_liste)
int *wnr_liste;
int anzahl;
int *erg_anzahl;
int **erg_liste;
```

wnr_liste ist ein Feld von Wortnummern, das die Eingabewortkette darstellt. *anzahl* ist die Anzahl der Wortnummern in diesem Feld. *erg_liste* ist die Menge der Wörter, die zurückgeliefert wird, und *erg_anzahl* gibt die Anzahl der Wortnummern in dieser Menge an.

Man gewinnt *yyparse_mod* durch einige Modifikationen aus dem BISON–Parser *yyparse*(). Die Funktion *yylex*() liest nicht mehr von einem Eingabeband, sondern liefert aus dem Feld *wnr_liste* das jeweils nächste Nichtterminalsymbol. Die in diesem Feld enthaltene Wortkette wird also wie bei der Syntaxanalyse üblich, analysiert. Wenn die Kette fertig bearbeitet ist, befindet sich *yyparse_mod* in einem von der Wortkette abhängigen Zustand Z. Aus der Analyseaktionstabelle kann dann abgelesen werden, mit welchen Wörtern die Wortfolge fortgesetzt werden kann. Jedes Terminalsymbol, also jede Wortnummer, bei dem im Zustand Z gemäß der Tabelle eine *shift*– oder eine *reduce*–Aktion ausgeführt wird, repräsentiert eines der gesuchten Wörter. Alle solchen Wörter werden im Feld *erg_liste* zusammengefaßt und zurückgeliefert. Wortnummern, bei denen in der Aktionstabelle im Zustand Z die Fehleraktion *error* vermerkt ist, sind nicht zulässig für die Erweiterung der Eingabekette.

Da die Tabellen des BISON–Parsers aus Komplexitätsgründen nicht in übersichtlicher Form dargestellt sind und nur durch zeitraubende Umwege die nötige Information herausgelesen werden kann, wurde folgender zusätzlicher Vektor als Konstante eingeführt:

$$yyerlaubte\_symbole[m][n]$$

yyerlaubte_symbole ist ein Feld mit m Listen von Terminalsymbolen. m ist die Anzahl der Zustände des Parsers. Jede der m Listen enthält alle Terminalsymbole, mit denen im zugeordneten Zustand die Eingabekette fortgesetzt werden darf. Der Vektor

$$yynerlsym[m]$$

enthält zu jedem Zustand die Anzahl der erlaubten Symbole. Die Größe n des Felds yyerlaubte_symbole[m][n] ist dann das Maximum dieser Größenangaben.

Die Einträge dieser Felder berechnet der BISON–Compilergenarator bei der Bestimmung der Analyseaktionstabellen. Die automatisch erzeugte Prozedur *yyparse_mod* muß nur noch aus diesen Strukturen die nötigen Informationen herauslesen und nicht mehr berechnen.

Beispiel: Der modifizierte Compilergenerator produziert aus der Beispielgrammatik G_{bsp}:

```
SY_ZEIT_SES
                : SY_ADV_1                  (1)
                | SY_ADV_3 SY_ADV_1         (2)
                ;
SY_ADV_1
                : H_WORTHYP_2               (3)
                ;
H_WORTHYP_2
```

Goto-Tabelle

	SY_ZEIT_SES	*SY_ADV_1*	*H_WORTHYP_2*	*SY_ADV_3*	*H_WORTHYP_4*	3851	375	378
0	1	2	4	3	5	6	7	8
1	—	—	—	—	—	—	—	—
2	—	—	—	—	—	—	—	—
3	—	9	4	—	—	10	7	11
4	—	—	—	—	—	—	—	—
5	—	—	—	—	—	—	—	—
6	—	—	—	—	—	—	—	—
7	—	—	—	—	—	—	—	—
8	—	—	—	—	—	—	—	—
9	—	—	—	—	—	—	—	—
10	—	—	—	—	—	—	—	—
11	—	—	—	—	—	—	—	—

Analyseaktions-Tabelle

	3851	375	378	$
0	shift	shift	shift	error
1	error	error	error	accept
2	error	error	error	reduce 1
3	shift	shift	shift	error
4	error	error	error	reduce 3
5	reduce 7	reduce 7	reduce 7	error
6	reduce 8	reduce 8	reduce 8	reduce 4
7	error	error	error	reduce 5
8	reduce 9	reduce 9	reduce 9	reduce 6
9	error	error	error	reduce 2
10	error	error	error	reduce 4
11	error	error	error	reduce 6

Abbildung 3.9: Analysetabellen des BISON–Parsers für die Beispielgrammatik G_{bsp}

```
                : 3851                  (4)
                | 375                   (5)
                | 378                   (6)
                ;
SY_ADV_3
                : H_WORTHYP_4           (7)
                ;
H_WORTHYP_4
                : 3851                  (8)
                | 378                   (9)
                ;
```

eine Prozedur *yyparse_mod* die mit den Analysetabellen in Abbildung 3.9 und den Vektoren in Abbildung 3.10 arbeitet.

yyerlaubte_symbole:

Zustand	erlaubte Symbole			
0	3851	375	378	
1	$			
2	$			
3	3851	375	378	
4	$			
5	3851	375	378	
6	3851	375	378	$
7	$			
8	3851	375	378	$
9	$			
10	$			
11	$			

yynerlsym:

0	1	2	3	4	5	6	7	8	9	10	11
3	1	1	3	1	3	4	1	4	1	1	1

Abbildung 3.10: Die Vektoren *yyerlaubte_symbole* und *yynerlsym* des modifizierten BISON–Parsers für die Beispielgrammatik G_{bsp}

3.2 Automatisches Generieren von Konstituentenhypothesen

Im Rahmen dieses Berichts wird davon ausgegangen, daß im Sprachsignal gezielt nach syntaktisch korrekten Wortfolgen gesucht wird, um Konstituentenhypothesen zu erzeugen. Die erlaubten Wortfolgen werden von links nach rechts aufgebaut und, ausgehend von einem bestimmten Startpunkt mit dem Sprachsignal verglichen, das heißt, sie werden verifiziert. Bei diesem Vergleich erhält man ein Bewertungsmaß dafür, wie gut die vorgegebene Wortfolge mit dem Sprachsignal übereinstimmt. Gesucht sind Wortketten, die möglichst gut bewertet sind, also möglichst gut mit dem Sprachsignal übereinstimmen.

Betrachtet man die erzeugten Wortketten als Zustände in einem Zustandsraum, entspricht der Algorithmus zum Generieren von Konstituentenhypoyhesen einer Suche nach dem bestbewerteten Zustand in diesem Zustandsraum. Die Suche erfolgt hier mit Hilfe des A^*–Algorithmus, der im Abschnitt 3.2.1 näher erläutert wird. Für den schrittweisen Aufbau der korrekten Wortketten wird dabei der modifizierte BISON–Parser verwendet, der im Abschnitt 3.1.2 beschrieben wurde.

Die Verifkation der Wortfolgen, also der Vergleich mit dem Sprachsignal, erfolgt mit Hilfe von Hidden–Markov–Modellen. In Abschnitt 3.2.2 wird der Ablauf dieser Verifikation erläutert. Aus den bestbewerteten vollständigen Wortfolgen werden Konstituentenhypothesen erzeugt. Der ganze Algorithmus zum automatischen Generieren von Konstituentenhypothesen ist in Abschnitt 3.2.3 dargelegt.

3.2.1 Der A^*–Algorithmus

Der A^*–Algorithmus (siehe [Nils71] und [Niem89]) arbeitet auf einer endlichen Menge von Zuständen. Während der Bearbeitung dieses Algorithmus wird ein gerichteter Graph, der Suchgraph, erzeugt. Die Knoten dieses Graphen entsprechen den Zuständen des Zustandraums und die Kanten den verschiedenen Übergängen von einem Zustand zum nächsten. Der Aufbau des Suchgraphen beginnt mit einer Menge von ausgezeichneten Startzuständen. Neue Knoten werden erzeugt, indem auf einen noch nicht bearbeiteten Knoten alle möglichen Übergänge angewendet werden. Man sagt, der Knoten wird ex-

pandiert. Ausgehend von den Startzuständen werden durch Expansion solange neue Knoten erzeugt, bis einer der vorher definierten Endzustände erreicht wird oder keine neuen Knoten mehr generiert werden können.

Würden die Knoten in der Reihenfolge expandiert werden, in der sie erzeugt werden, könnte in Folge einer kombinatorischen Explosion des Suchgraphen der Aufwand für die Suche zu groß werden. Dasselbe gilt, wenn immer der zuletzt generierte Knoten bearbeitet wird. Um daher aus der Menge aller noch nicht bearbeiteten Knoten möglichst effizient einen auszuwählen, der als nächstes expandiert werden soll, werden jedem Knoten des Suchbaums Kosten zugeordnet. Für jeden Knoten v_i wird eine Kostenfunktion $\varphi(v_i)$ definiert, die die Kosten des optimalen Pfades durch den Suchbaum angibt, der von einem Startknoten v_0 über den Knoten v_i zu einem von v_i erreichbaren Endknoten v_g führt. Ein solcher Pfad stellt eine Lösung des Algorithmus dar. Die Aufgabe des A^*–Algorithmus ist, den optimalen Lösungspfad zu finden. Die Kosten dieses Pfades setzen sich aus zwei Bestandteilen zusammen, zum einen den minimalen Kosten $\psi(v_i)$ von v_0 zu v_i und zum anderen den minimalen Kosten $\chi(v_i)$ von v_i nach v_g. Es gilt somit (siehe [Niem89]):

$$\varphi(v_i) = \psi(v_i) + \chi(v_i)$$

Da die wahren Kosten im allgemeinen nicht bekannt sind, werden tatsächlich Schätzungen dieser Kosten verwendet, für die ebenfalls gilt:

$$\hat{\varphi}(v_i) = \hat{\psi}(v_i) + \hat{\chi}(v_i)$$

Um zu gewährleisten, daß der A^*–Algorithmus immer einen optimalen Pfad zu einem Zielknoten findet, vorausgesetzt daß einer vorhanden ist, muß für die Restfunktion $\chi(v_i)$ eine optimistische Abschätzung $\hat{\chi}(v_i)$ gefunden werden, das heißt:

$$\hat{\chi}(v_i) \leq \chi(v_i)$$

Zusätzlich gilt, daß ein A^*–Algorithmus niemals mehr Knoten expandiert als ein anderer, wenn er eine Schätzfunktion $\hat{\chi}_1$ verwendet, die immer größer ist als die Schätzfunktion $\hat{\chi}_2$ des anderen Algorithmus, also:

$$\forall v: \qquad \hat{\chi}_1(v) > \hat{\chi}_2(v)$$

Der A^*–Algorithmus arbeitet mit zwei Listen: Die Liste OFFEN enthält alle Knoten, die als nächstes zur Bearbeitung anstehen. Die Liste GESCHLOSSEN nimmt die Knoten auf, die bereits abgearbeitet wurden. Der A^*–Algorithmus lautet dann folgendermaßen:

Eintragen der Startknoten in die Liste OFFEN
Berechnen aller Kosten $\hat{\varphi}$ der Startknoten
solange OFFEN nicht leer ist
    Entfernen des Knotens v_i mit den besten Kosten $\hat{\varphi}(v_i)$ aus der Liste OFFEN
    Eintragen von v_i in die Liste GESCHLOSSEN
    wenn v_i ein Zielknoten ist
    dann Halt: der Pfad von v_0 zu v_i ist der Lösugspfad, wobei v_0 ein Startknoten ist (Dieser Pfad wird durch Verfolgen der Rückwärtszeiger von v_i zu v_0 gefunden)
    sonst Expandieren von v_i
        Berechnen der Kosten $\hat{\varphi}$ der Nachfolger von v_i
        Setzen von Rückwärtszeigern von den Nachfolgern zu v_i
        Eintragen der Nachfolger in die Liste OFFEN, die weder in der Liste OFFEN noch in der Liste GESCHLOSSEN enthalten waren
        wenn einer der Nachfolger v_{ij} bereits in der Liste OFFEN enthalten war
        dann wenn die neu berechneten Kosten $\hat{\varphi}(v_{ij})$ besser als die vorher berechneten sind
            dann Streichen des alten Rückwärtszeigers von v_{ij}
            sonst Streichen des neuen Rückwärtszeigers von v_{ij} nach v_i
Halt: kein Lösungsweg gefunden.

3.2.2 Verifikation von Wortketten mit Hidden–Markov–Modellen

Um ein Bewertungsmaß für die Ähnlichkeit zwischen Wortketten und Sprachsignal zu bekommen, werden die Wortketten mit Hilfe eines statistischen Verfahrens verifiziert, das auf Hidden–Markov–Modellen basiert. Dabei wird zu jeder Wortkette ein Referenzmodell in Form eines Hidden–Markov–Modells erzeugt, das dann mit dem Sprachsignal verglichen wird. Diese Referenzmodelle modellieren die Aussprache der einzelnen Wörter und werden automatisch aus der phonetischen Umschrift der Wörter im Lexikon gewonnen.

Den hier verwendeten Hidden–Markov–Modellen (siehe [Rabi89], [Niem89]) liegt eine endliche Menge von Zuständen

$$S = \{S_1, S_2, \ldots, S_I\}$$

zugrunde. Zu einem Zeitpunkt t_n befindet sich das Modell im Zustand $s_n \in S$. Zum Zeitpunkt t_{n+1} geht das Modell in den Zustand $s_{n+1} \in S$ über. In der Matrix

$$\mathbf{A} = [a_{ij}] = P(s_{n+1} = S_j \mid s_n = S_i) \qquad \text{wobei } i, j \in \{1, 2, \ldots I\}$$

werden die Wahrscheinlichkeiten für solche Übergänge zusammengefaßt. Die Werte a_{ij} sind unabhängig von der Zeit und den Zuständen, die vorher eingenommen worden sind. Ein Zustand aus der Menge S wird mit einer Wahrscheinlichkeit als Startzustand s_1 ausgewählt, die durch den Vektor

$$\mathbf{\Pi} = [\pi_i] = P(s_1 = S_i) \qquad \text{mit } i \in \{1, 2, \ldots I\}$$

gegeben ist. Die Zustände eines Hidden–Markov–Modells selbst sind nicht beobachtbar. Ein sichtbares Bestandteil von Hidden–Markov–Modellen ist eine endliche Menge von Ausgabesymbolen

$$O = \{O_1, O_2, \ldots O_L\},$$

aus der eine Folge $\mathbf{o} = [o_i]$ erzeugt wird. Ein Symbol $o_i \in O$ wird erzeugt, wenn sich das Modell zum Zeitpunkt t_n im Zustand s_i befindet und gemäß der Ausgabewahrscheinlichkeiten

$$\mathbf{B} = [b_{il}] = P(O_l \text{ wird im Zustand } S_i \text{ ausgegeben} \mid s_n = S_i)$$
$$\text{wobei } i \in \{1, \ldots I\},\ l \in \{1, \ldots L\}$$

das Ausgabesymbol $o_i = O_l$ ausgewählt wird. Die Matrix $\mathbf{B}$ gibt die diskrete Verteilung der Dichtefunktion $b_i(l)$ an. Ein Hidden–Markov–Modell HMM ist durch das Tripel

$$HMM = (\mathbf{\Pi}, \mathbf{A}, \mathbf{B})$$

eindeutig beschrieben. Eine Folge von Ausgabesymbolen $\mathbf{o} = (o_1, o_2, \ldots o_T)$ wird wie folgt generiert: Zunächst wird ein Zustand S_i mit der Wahrscheinlichkeit π_i als Startzustand s_1 ausgewählt. Dann geht das Modell mit der Wahrscheinlichkeit a_{ij} zum Zustand S_j über und liefert mit der Wahrscheinlichkeit b_{il} das Ausgabesymbol $O_l = o_1$. Das Hidden–Markov–Modell befindet sich dann zum Zeitpunkt t_2 im Zustand s_2. Übergänge und das Erzeugen von Ausgabesymbolen wird solange fortgeführt, bis die Kette $\mathbf{o}$ mit der Länge T generiert worden ist.

Die hier verwendeten Referenzmodelle werden aus den phonetischen Umschriften der Wörter beziehungsweise Wortketten aus einem Lexikon konstruiert. Die Aussprachebeschreibungen werden in Laute gegliedert. Jeder Laut wird durch ein Hidden–Markov–Modell modelliert, das aus zwei Zuständen und den drei folgenden Übergängen besteht:

- SUB: Ersetzen eines Lauts des Referenzwortes durch einen des Sprachsignals;
- DEL: Löschung eines Lauts des Referenzwortes;
- INS: Einfügen eines zusätzlichen Lauts;

Die Referenzmodelle sind durch sogenannte **Links–Rechts–Modelle** oder **Bakis–Modelle** dargestellt. Solche Modelle sind dadurch gekennzeichnet, daß beim Übergang vom Zustand $s_n = S_{i_n}$ zum Zustand $s_{n+1} = S_{i_{n+1}}$ gilt: $i_n \leq i_{n+1}$ mit i_n, $i_{n+1} \in \{1, \ldots I\}$. Die Zustandsübergänge führen von links nach rechts.

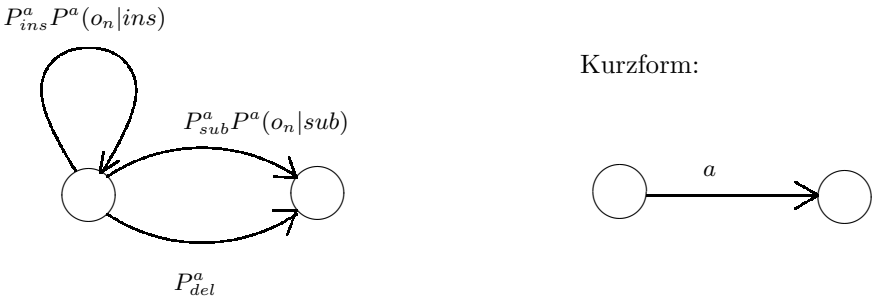


Abbildung 3.11: Hidden–Markov–Modell, das den Laut 'a' modelliert

Abbildung 3.11 zeigt als Beispiel den Graphen eines Hidden–Markov–Modells, das den Laut 'a' modelliert. Die Kreise entsprechen dabei den Zuständen und die Pfeile stellen die Übergänge dar. P^a_x ist die Wahrscheinlichkeit, daß die Kante $x \in \{\text{SUB}, \text{DEL}, \text{INS}\}$ abgearbeitet wird. $P^a(o_n|x)$ ist die bedingte Wahrscheinlichkeit dafür, daß bei Verfolgung der Kante $x \in \{\text{SUB}, \text{INS}\}$ das Segment o_n des Sprachsignals ausgegeben wird. Das Sprachsignal ist in Segmente unterteilt, wobei jedes Segment nach Lautklassen klassifiziert worden ist.

Die Modellparameter a_{ij} der Hidden–Markov–Modelle werden mit dem Baum–Welch–Algorithmus trainiert (siehe [Kuhn87], [Rabi89], [LeRS83]).

Die Wahrscheinlichkeit $P(\mathbf{o} \mid HMM)$, daß von einem Hidden–Markov–Modell die Segmentfolge $\mathbf{o}$ produziert wird, kann folgendermaßen berechnet werden: Die Vorwärtswahrscheinlichkeit

$$\alpha_{ni} = P(o_1, \ldots o_n, \;\; s_n = S_i \mid HMM)$$

gibt die Wahrscheinlichkeit an , daß zum Zeitpunkt t_n die ersten n Symbole von $\mathbf{o}$ erzeugt worden sind und das Hidden–Markov–Modell HMM sich im Zustand S_i befindet. α wird mit

$$\alpha_{1i} = \pi_i b_{il} \qquad \text{wobei } \; O_l = o_1, \;\; i \in \{1, \ldots I\}$$

initialisiert. Die Werte von α können dann rekursiv berechnet werden:

$$\alpha_{n+1,j} = (\sum_{i=1}^{I} \alpha_{ni}\alpha_{ij})b_{jl} \qquad \text{mit } \; n \in \{1, \ldots T-1\}, \;\; j \in \{1, \ldots I\}, \;\; O_l = o_{n+1}$$

Dann ist

$$P(\mathbf{o}|HMM) = \sum_{i=1}^{I} \alpha_{Ti}$$

3.2.3 Algorithmus zum Erzeugen von Konstituentenhypothesen

Mit Hilfe des A^*–Algorithmus werden die bestbewerteten syntaktisch korrekten Wortketten im Sprachsignal gesucht, aus denen dann Konstituentenhypothesen erzeugt werden. Zu diesem Zweck werden den Suchgraphknoten des A^*–Algorithmus Wortfolgen mit dem jeweiligen Anfangs– und Endpunkt im Sprachsignal zugeordnet. Das Anhängen einer weiteren Wortnummer an eine solche Wortfolge stellt eine Kante von einem Knoten zum nächsten dar. Um einen Knoten zu expandieren, werden alle Wörter ermittelt, mit denen die Wortkette, die dem Knoten zugrundeliegt, syntaktisch korrekt ergänzt werden kann. Hierfür wird der modifizierte BISON–Parser aus Abschnitt 3.1.2 verwendet. Die Nachfolgeknoten des expandierten Knotens repräsentieren dann die auf diese Weise gebildeten Wortketten. Da beim schrittweisen Aufbau der möglichen Wortfolgen kein Knoten mehrfach erzeugt wird, bildet der Suchgraph einen Baum. Die Wurzel dieses Baums entspricht der leeren Wortkette. Der Baum wird mit ein oder mehreren Startknoten initialisiert, die jeweils durch eine Kante mit der Wurzel verbunden sind. Jeder Startknoten wird aus der Kombination eines Startsegments und eines Worts gebildet, das einen gültigen Anfang einer syntaktisch korrekten Wortkette darstellt. Jedes der Segmente, aus denen das zu

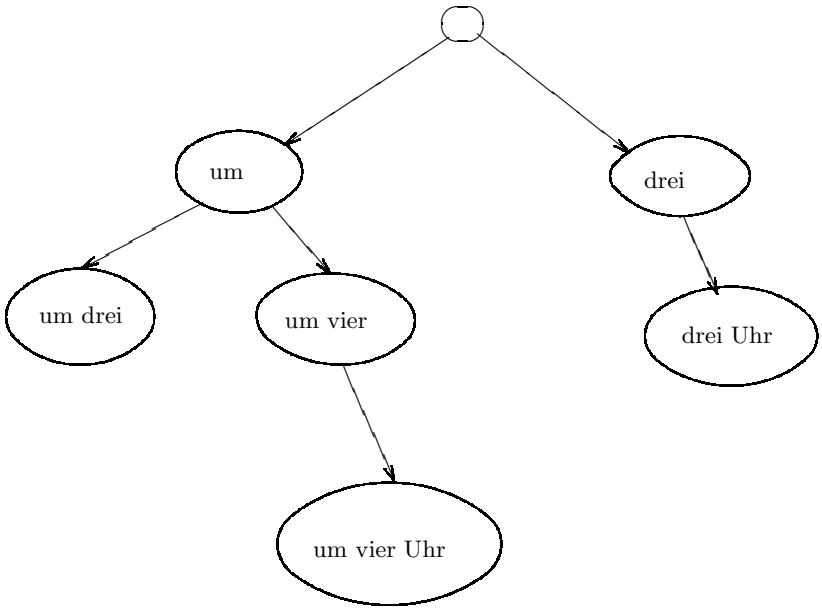


Abbildung 3.12: Ausschnitt aus einem Suchbaum für die Zeitangaben

untersuchende Sprachsignal aufgebaut ist, kann dabei als Startsegment verwendet werden. Alle Wortketten, die syntaktisch vollständig sind, bilden einen Endknoten im Suchbaum. Gesucht wird nach den bestbewerteten Endknoten. Abbildung 3.12 zeigt als Beispiel einen Suchbaum, dessen Knoten mit den entsprechenden Wortketten gekennzeichnet sind.

Den einzelnen Knoten des Suchbaums müssen Kosten zugewiesen werden. Ein Maß für diese Kosten liefert die Ähnlichkeit zwischen der Wortkette, die dem Knoten zugeordnet ist und der Äußerung im Sprachsignal. Bei der Verifikation der Wortketten erhält man eine Bewertung dieser Ähnlichkeit, die zur Bestimmung der Kosten herangezogen werden kann. Es ist dabei zu beachten, daß gilt: "je größer die Bewertung der Wortkette, desto besser ihre Güte". Da für einen Suchbaumknoten gilt "je geringer die Kosten des Knotens, desto besser seine Güte", können die Bewertungen der Wortketten nicht direkt mit den Kosten der Suchbaumknoten identifiziert werden. Für die Bestimmung der Kosten eines Suchbaumknotens wird zur entsprechenden Wortkette ein Hidden–Markov–Modell (HMM) generiert und die Ähnlichkeit

$$P(o_a \dots o_e \mid HMM(\text{Wortkette})),$$

zwischen Modell und dem segmentierten Sprachsignal bestimmt. o_a und o_e sind dem ersten beziehungsweise letzten Segment des zu berechnenden Bereichs des Sprachsignals gleichzusetzen. a beziehungsweise e geben die jeweilige Segmentnummer an. Die Kosten des Suchbaumknotens ergeben sich schließlich aus dem negativen Logarithmus von P.

Zu Beginn der Suche werden die einzelnen Wörter der Startknoten verifiziert. Zur Berechnung der Ähnlichkeit P wird hierbei der sogenannte Profilvektor definiert. Die Elemente dieses Vektors geben die Ähnlichkeiten für alternative Endpunkte des gesuchten Wortes an. Der Bereich des Sprachsignals, mit dem verglichen wird, reicht von o_a bis o_{a+k}. Da die Länge des Wortes im Sprachsignal nicht bekannt ist, wird diese mit der Konstante k festgesetzt. Der Profilvektor entspricht der letzten Zeile in der vom Referenzmodell (Wortkette) und Sprachsignal (Segmentfolge) aufgespannten α–Matrix, die die Vorwärtswahrscheinlichkeiten angibt. Das Segment, das dem Maximum aller Elemente des Profilvektors zugeordnet ist, wird als Endpunkt o_{e1} im Sprachsignal für das Wort festgelegt.

Bei der Expansion eines Suchbaumknotens wird eine gegebene Wortfolge erweitert. Um die Ähnlichkeit zwischen der neuen Wortfolge und dem Sprachsignal zu bestimmen, müßte die gesamte α–Matrix berechnet werden, die von dieser Wortfolge und dem entsprechenden Teil des Sprachsignals aufgespannt wird. Die Verifikation der neuen Wortfolge wird nun optimiert, indem nur der Teil der Matrix berechnet wird, der dem neu hinzugefügten Wort und dem Signalausschnitt von o_{e1} bis o_{e1+k} entspricht. Die Ähnlichkeitsberechnung wird dabei mit dem Profilvektor der Teilfolge initialisiert, die

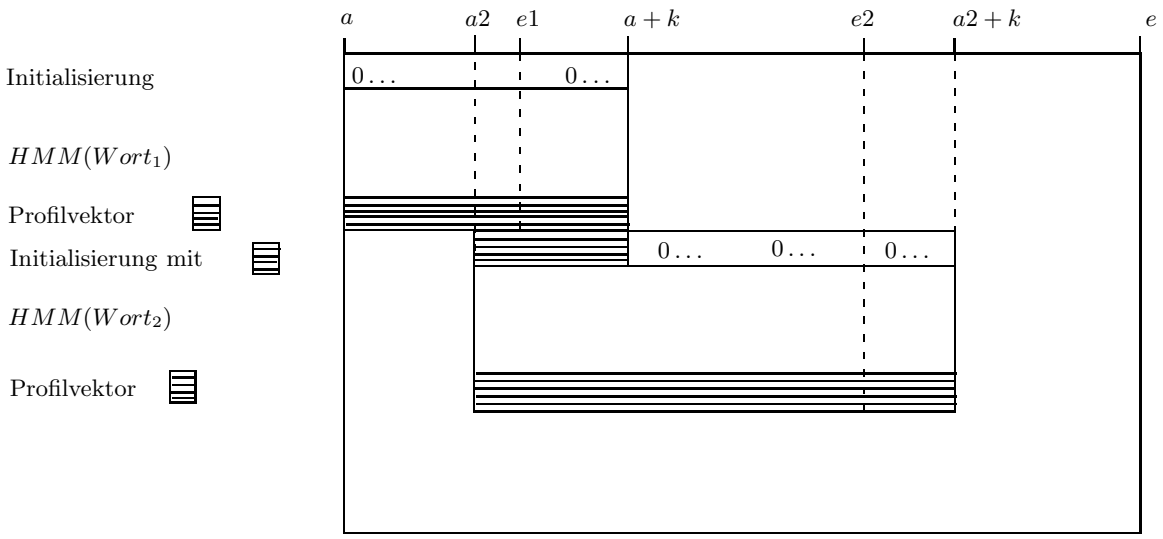


Abbildung 3.13: Optimierung der Ähnlichkeitsberechnung bei der Erweiterung einer Wortkette. Dargestellt ist die vom Referenzmodell und Sprachsignal aufgespannten α–Matrix

um das neue Wort erweitert werden soll. Weil durch Verschleifungen über Wortgrenzen hinweg die Grenze zwischen zwei Wörtern nicht genau festgelegt werden kann, beginnt die Ähnlichkeitsberechnung des neuen Wortes nicht bei o_{e1}, sondern schon bei o_{e1-x}. Zu jedem Knoten im Suchbaum muß also der entsprechende Profilvektor aufgehoben werden. Abbildung 3.13 stellt die Optimierung der Ähnlichketsberechnung graphisch dar.

Jeder Suchbaumknoten erhält die Wortnummer des hinzugefügten Wortes, das Anfangs- beziehungsweise das Endsegment, den Profilvektor und eine Pfadrückverfolgung. Ein Zielknoten des Suchbaums schließlich ist dadurch gekennzeichnet, daß ihm eine Wortkette zugeordnet ist, die syntaktisch vollständig ist. Wenn der bestbewertete Knoten in der Menge OFFEN ein Zielknoten ist, wird aus diesem eine Konstituentenhypothese erzeugt, die dann für die weitere Bearbeitung zur Verfügung steht. Eine Konstituentenhypothese besteht aus einer identifizierenden Kettennummer, einem Anfangssegment, einem Endsegment, einer Bewertung und der entsprechenden Folge von Wortnummern. Diese Angaben werden entsprechend aus dem Lösungspfad im Suchbaum ermittelt. Der Algorithmus endet erst, wenn die Menge OFFEN leer ist. Auf diese Weise werden Konstituentenhypothesen, der Bewertung nach geordnet, erzeugt.

In der Abbildung 3.14 ist der Ablauf des ganzen Algorithmus in einem Struktogramm dargestellt. Da nur ein Suchbaum, nicht ein Suchgraph bei der Analyse des Sprachsignals aufgebaut wird, kann auf die Menge GESCHLOSSEN verzichtet werden.

```
Lexika einlesen
Sprachsignal einlesen
Liste OFFEN initialisieren
FOR alle Segmente des Sprachsignals
    FOR alle Wörter, mit denen syntaktisch korrekte Wortketten begonnen werden
        können
        Wort verifizieren
        Suchbaumknoten erzeugen
        Wortnummer, Bewertung, Anfang, Ende, Profilvektor und Pfadrückverfol-
        gung in den Knoten eintragen
        Knoten in die Menge OFFEN eintragen
WHILE die Menge OFFEN nicht leer ist
    den bestbewerteten Knoten aus der Menge OFFEN entfernen
    IF dieser Knoten ein Zielknoten ist
    THEN Konstituentenhypothese generieren
    ELSE FOR alle Wörter, mit denen syntaktisch korrekt die Wortkette dieses
             Knotens fortgesetzt werden kann
         Wortkette verifizieren
         Suchbaumknoten erzeugen
         Wortnummer, Bewertung, Anfang, Ende, Profilvektor und Pfadrück-
         verfolgung in den Knoten eintragen
         Knoten in die Menge OFFEN eintragen
```

Abbildung 3.14: Algorithmus zum Generieren von Konstituentenhypothesen

Kapitel 4

Ergebnisse

Der Teil der EVAR–Wissensbasis, der Zeitangaben repräsentiert, wurde in Kapitel 2.2 vorgestellt. Es handelt sich um einen ERNEST–Netzausschnitt, dessen Wurzel das Konzept SY_ZEITKETTE ist. Jedes Konzept des Netzes besitzt Bestandteile und Konkretisierungen, mit denen es durch die jeweiligen Kanten verbunden ist. Die unterste Ebene wird aus aus dem Konzept H_WORTHYP gebildet, das aus der Hypothesenebene der Wissensbasis stammt.

Um aus dem ERNEST–Netz eine kontextfreie Grammatik zu extrahieren, müssen Informationen verarbeitet werden, die in den Kantenbeschreibungen und in den Modalitätsbeschreibungen der einzelnen Konzepte enthalten sind (siehe Kapitel 2.3). Nicht alle nötigen Information sind explizit im ERNEST–Netz vermerkt, sondern werden erst bei der Analysekontrolle berechnet. Um diese Informationen zu erhalten, wird ein Suchbaum erzeugt, dessen Knoten ausschließlich modifizierte Konzepte enthalten. Dies wird durch Modifikationen am ERNEST–Kontrollalgorithmus erreicht. Auf das Zielkonzept SY_ZEITKETTE wird zuerst Regel 6 angewendet, das heißt es wird zum Zielkonzept ein modifiziertes Konzept erzeugt. Anschließend wird nur noch Regel 5 ausgeführt und solange zu Bestandteilen und Konkretisierungen bereits erzeugter modifizierter Konzepte neue modifizierte Konzepte generiert, bis alle Konzepte und Kanten bis in die unterste Hierarchiestufe des Netzausschnitts für Zeitangaben bearbeitet worden sind. Jedesmal, wenn eine Bestandteils– oder Konkretisierungskante einer bestimmten Modalitätsbeschreibung bearbeitet wird, wird ein neuer Suchbaumknoten erzeugt, der um ein modifiziertes Konzept erweitert ist. Bei der Expansion eines Suchbaumknotens werden auf diese Weise alle Nachfolgerknoten berechnet.

Da alle obligatorischen, optionalen und inhärenten Bestandteile und Konkretisierungen berücksichtigt werden, entstehen sehr viele Suchbaumknoten. Hier entstand folgendes Problem: Der zu berechnende Suchbaum nahm Ausmaße an, die die Speicherkapazität der verwendeten Rechner sprengte. Es wurde ein virtueller Speicher von 120 Mbyte eingerichtet. Für die Berechnung des erforderlichen Suchbaums wäre aber ein Vielfaches dieser Kapazitäten nötig gewesen. Um dennoch eine Grammatik zu erhalten wurde folgender Weg beschritten: Ausgehend vom Zielkonzept SY_ZEITKETTE werden in einem ersten Schritt Suchbaumknoten solange expandiert, bis alle Konzepte bis zu einem vorgegebenen Niveau innerhalb der Netzhierarchie modifiziert worden sind. Gewählt wurden folgende Konzepte, zu denen in diesem ersten Schritt modifizierte Konzepte generiert werden:

```
SY_ZEITKETTE
SY_TYP_ANGABE
SY_TYP_SPANNE
SY_TYP_DATUM
SY_ZEIT_DAU
SY_ZEIT_DTG
SY_ZEIT_ITV
SY_ZEIT_KTX
SY_ZEIT_SES
SY_ZEIT_TAG
SY_ZEIT_TDZ
SY_ZEIT_TZT
SY_ZEIT_UHR
SY_ZEIT_WOC
SY_ZEIT_WUHR
SY_ZEIT_ZEH
```

Diese Konzepte bilden den oberen Teil des Ausschnitts aus der Wissensbasis, der Zeitangaben repräsentiert. Aus diesem Teil wird wie oben beschrieben, ein Suchbaum erzeugt. Aus den Blattknoten werden Grammatikregeln produziert (siehe Kapitel 2.3). Da hier nicht bis in die Hypothesenebene expandiert wird, werden auch keine Regeln erzeugt, die Terminalsymbole enthalten. Dies geschieht im nächsten Schritt.

Zu jedem Konzept, zu dessen Bestandteilen und Konkretisierungen noch keine modifizierten Konzepte generiert worden sind, wird im zweiten Schritt ein eigener Suchbaum berechnet. Aus jedem dieser Suchbäume wird eine eigene vollständige Grammatik extrahiert, die genau die Wortfolgen erkennt, die das jeweilige Konzept modelliert.

Um alle Grammatiken, die im zweiten Schritt erzeugt werden, und die Grammatikregeln, die im ersten Schritt entstehen, zu einer Grammatik zu vereinen, die alle Zeitangaben repräsentiert, muß folgendes beachtet werden: Jedes neu erzeugte modifizierte Konzept erhält während der Suchbaumberechnung eine eindeutige Identifikation. Da jeder einzelne Suchbaum unabhängig von den anderen generiert wird, können modifizierte Konzepte, die zwei verschiedenen Suchbäumen angehören, dieselbe Identifikation haben. Dies führt zu Schwierigkeiten, weil diese Identifikationen dazu verwendet werden, Nichtterminalsymbole, die aus modifizierten Konzepten erzeugt werden, zu unterscheiden. Ein Nichtterminalsymbol ist aufgebaut aus einem Konzeptnamen und einer angefügten Nummer. Diese Nummer kann nun nicht mehr mit der Identifikation des entsprechenden modifizierten Konzepts gleichgesetzt werden. Zu jeder Identifikation wird eine Konstante hinzugezählt, die abhängig von dem Suchbaum ist, zu dem das entsprechende modifizierte Konzept gehört.

Beispiel: Bei der Berechnung eines Suchbaums zum Konzept SY_ZEIT_UHR wird ein modifiziertes Konzept zu SY_NOMEN mit der Identifikation 3 berechnet. Bei der Berechnung eines Suchbaums zum Konzept SY_ZEIT_WUHR wird ebenfalls ein modifiziertes Konzept zu SY_NOMEN mit der Identifikation 3 generiert. Dem ersten Suchbaum wird die Konstante 3000 und dem zweiten die Konstante 6000 zugeordnet. Im ersten Fall wird aus dem modifizierten Konzept zu SY_NOMEN das Nichtterminalsymbol SY_NOMEN_3003 und im zweiten Fall das Nichtterminalsymbol SY_NOMEN_6003 berechnet.

Auf diese Weise können die Grammatiken, die im zweiten Schritt berechnet werden, ohne Schwierigkeiten vereinigt werden. Anders verhält es sich mit den Grammatikregeln, die im ersten Schritt erzeugt werden. Diese Regeln enthalten Nichtterminalsymbole in den rechten Regelseiten, für die es keine Ersetzungsregeln gibt, die also nicht die linke Seite einer Grammatikregel bilden. Wie ein solches Nichtterminalsymbol zu ersetzen ist, schreibt aber eine der Grammatiken vor, die im zweiten Schritt generiert werden.

Beispiel: Im ersten Schritt wird folgende Regel generiert:

```
SY_ZEITKETTE_0 : SY_ZEIT_KTX_1 SY_TYP_ANGABE_2 ;
```

Da zum Konzept SY_ZEIT_KTX im ersten Schritt keine Bestandteile und Konkretisierungen berechnet worden sind, gibt es keine Regel mit dem Nichtterminalsymbol SY_ZEIT_KTX_1 auf der linken Seite. Im zeiten Schritt wird für das Konzept SY_ZEIT_KTX ein Suchbaum aufgebaut und aus diesem eine kontexfreie Grammatik extrahiert. Das Nichtterminalsymbol SY_ZEIT_KTX_0 stellt das Startsymbol dieser Grammatik dar, für das es auch eine Ersetzungsregel gibt.

Die Grammatikregeln aus dem ersten Schritt und die Grammatikregeln aus dem zweiten Schritt werden wie folgt modifiziert: Den Nichtterminalsymbolen der Regeln aus dem ersten Schritt werden keine Nummern angehängt. Sie sind mit den Namen der entsprechenden modifizierten Konzepte identisch. Das gleiche gilt für die Startsymbole der Grammatiken, die im zweiten Schritt erzeugt werden. Das Startsymbol der so konstruierten vollständigen Grammatik ist dann SY_ZEITKETTE. Die so modifizierten Grammatikregeln werden nun zu einer vollständigen Grammatik vereinigt, die alle Wortfolgen erkennt, die Zeitangaben repräsentieren. Im Anhang B ist diese Grammatik aufgelistet.

Bei der modellgetriebenen Generierung modifizierter Konzepte wird der Gültigkeitsbereich der Konzepte gemäß der Kantenrestriktionen eingeschränkt. Die einzelnen Wörter, auf die der Gültigkeitsbereich eines Konzepts begrenzt ist, sind dann im Analyseparameter '*wort_nr*' abgelegt. Besitzt dieses Konzept eine Konkretisierungskante mit dem Zielkonzept H_WORTHYP und wird durch Anwendung

der Regel 5 zu H_WORTHYP über diese Konkretisierungskante ein modifiziertes Konzept erzeugt, werden die Wortnummern in das Attribut '*anforderung*' diese modifizierten Konzepts eingetragen. Die Einschränkung wird also weitergereicht. Die Menge der Wörter, auf die der Gültigkeitsbereich eines Konzepts eingeschränkt ist, wird nun nicht immer in Form von Einzelwerten im Analyseparameter '*wort_nr*' abgelegt, sondern kann auch in Form eines Intervalls festgelegt sein. Zum Beispiel stellen die den Ziffern 1 bis 10 entsprechenden Wortnummern ein zusammenhängendes Intervall im Definitionsbereich der Wortnummern dar. Wird die Menge von Wörtern, auf die ein Konzept eingeschränkt ist, in Form eines Intervalls gespeichert, wird er nicht wie die Einzelwerte im Analyseparameter '*wort_nr*' abgelegt, sondern im Analyseparameter '*zahlwert*'. Ein solches Intervall wird bei Anwendung der Regel 5 aber nicht über eine möglicherweise vorhandene Konkretisierungskante an ein modifiziertes Konzept von H_WORTHYP weitergereicht. Durch diesen Fehler in der Wissensbasis werden Grammatikregeln folgender Form erzeugt, da das Attribut '*anforderung*' des entsprechenden modifizierten Konzepts zu H_WORTHYP keinen Eintrag enthält.

```
H_WORTHYP_3     :
                ;
```

Das Nichtterminalsymbol H_WORTHYP_3 wird in diesem Fall durch die leere Symbolfolge ersetzt. Im ERNEST–Netz sind alle Zahlenangaben, die durch die Konzepte SY_ZAHLWORT und SY_ORDZAHL repräsentiert werden, in Form von Intervallen eingeschränkt. Das Konzept SY_ZAHLWORT ist Bestandteil des Konzepts SY_ZEIT_ZEH, das eine Zeiteinheit wie '*in fünf Minuten*' repräsentiert, und des Konzepts SY_SIM_UHR, das eine einfache Uhrzeit wie '*um zehn*' darstellt. Das Konzept SY_ORDZAHL ist Bestandteil des Konzepts SY_ZEIT_DAT, das ein Datum wie '*am dritten Oktober*' repräsentiert, und des Konzepts SY_ZEIT_TAG, das einen Wochentag modelliert. Durch den Umstand, daß Intervalle nicht bis in die Hypothesenebene weitergereicht werden, kann die extrahierte Grammatik keine Zahlenangaben der obigen Form erkennen.

Daten zu den erzeugten Suchbäumen und den daraus extrahierten Grammatiken finden sich in den Tabellen 4.1 und 4.2.

Konzept	Anzahl erzeugter modifizierter Konzepte	Anzahl erzeugter Knoten	benötigte CPU–Zeit in Sekunden
SY_TYP_SPANNE	512	513	19.03
SY_TYP_DATUM	1133	1134	89.45
SY_ZEIT_DAU	14	15	2.45
SY_ZEIT_DTG	14	15	2.38
SY_ZEIT_ITV	2	3	2.22
SY_ZEIT_KTX	12	13	2.25
SY_ZEIT_SES	4	5	2.28
SY_ZEIT_TAG	54	55	3.25
SY_ZEIT_TDZ	28	29	2.82
SY_ZEIT_TZT	38	39	3.15
SY_ZEIT_UHR	485	486	15.05
SY_ZEIT_WOC	66	67	4.18
SY_ZEIT_WUHR	8	9	2.47
SY_ZEIT_ZEH	48	49	3.05

Tabelle 4.1: Angaben zu den Suchbäumen, die zu den verschiedenen Konzepten erzeugt werden

Die aus den Suchbäumen extrahierten Grammatiken dienen als Eingabe des BISON–Compilergenerators. Dieser Compilergenerator wurde wie in Kapitel 3.1.2 modifiziert und erzeugt einen Parser, der zu einer vorgegeben Wortfolge diejenige Menge von Wörtern bestimmt, mit denen die Wortfolge syntaktisch korrekt fortgesetzt werden kann. Da die aus den Suchbäumen gewonnenen Grammatiken zwar kontexfrei, aber keine *LALR*–Grammatiken sind, treten bei der Parsererzeugung *reduce / reduce*– und *shift / reduce*–Konflikte auf. Tabelle 4.3 liefert dazu nähere Angaben.

Konzept	Anzahl der Regeln	Anzahl der Nichtterminalsymbole	Anzahl der Terminalsymbolde
SY_TYP_SPANNE	197	66	53
SY_TYP_DATUM	630	114	97
SY_ZEIT_DAU	22	11	12
SY_ZEIT_DTG	38	15	15
SY_ZEIT_ITV	5	3	3
SY_ZEIT_KTX	24	9	12
SY_ZEIT_SES	9	5	3
SY_ZEIT_TAG	159	39	40
SY_ZEIT_TDZ	67	27	29
SY_ZEIT_TZT	104	39	48
SY_ZEIT_UHR	158	49	36
SY_ZEIT_WOC	168	47	41
SY_ZEIT_WUHR	14	9	7
SY_ZEIT_ZEH	76	29	24
SY_ZEITKETTE	1820	465	167

Tabelle 4.2: Angaben zu den Grammatiken, die aus den verschiedenen Suchbäumen extrahiert werden

Startsymbol der Eingabegrammatik	*reduce / reduce*– Konflikte	*shift / reduce*– Konflikte
SY_TYP_SPANNE	63	9
SY_TYP_DATUM	1383	140
SY_ZEIT_DAU	2	—
SY_ZEIT_DTG	3	—
SY_ZEIT_ITV	—	—
SY_ZEIT_KTX	14	—
SY_ZEIT_SES	—	—
SY_ZEIT_TAG	8	108
SY_ZEIT_TDZ	5	—
SY_ZEIT_TZT	6	—
SY_ZEIT_UHR	29	13
SY_ZEIT_WOC	—	—
SY_ZEIT_WUHR	—	—
SY_ZEIT_ZEH	13	9
SY_ZEITKETTE	14665	3595

Tabelle 4.3: Angaben zu Konflikten, die bei der Parsergenerierung zu verschiedenen Eingabegrammatiken auftreten

Kapitel 5

Zusammenfassung

Das sprachverstehende Dialogsystem EVAR (**E**rkennen, **V**erstehen, **A**ntworten, **R**ückfragen), das am Lehrstuhl für Informatik 5 der Universität Erlangen–Nürnberg entwickelt wird, ist ein wissensbasiertes Analysesystem. Das für die Analyse des Sprachsignals notwendige linguistische Wissen ist in einer Wissensbasis in Form eines semantischen Netzes abgelegt. Das Netz ist im ERNEST–Formalismus (**Er**langer **Ne**tzwerk**sys**tem) repräsentiert, das ebenfalls am Lehrstuhl für Informatik 5 entwickelt wird. Es enthält explizit Wissen über Syntax und Semantik der deutschen Sprache. Gewählt wurde als Anwendungsgebiet der Themenbereich 'Intercity–Zugauskunft'. In der Wissensbasis des sprachverarbeitenden Systems, also im ERNEST–Netz, ist daher auch Wissen über diesen Anwendungsbereich und über den Dialogbereich abgelegt. Die Verarbeitung von EVAR gliedert sich in eine Erkennungs– und eine Verstehensphase.

Um Restriktionen über der Menge der möglichen Äußerungen, die explizit in der Wissensbasis aufgeführt sind, möglichst früh schon im Erkennungsprozeß einsetzen zu können, benötigt man ein Sprachmodell, das diese Restriktionen in einem Grammatikformalismus vereinigt, der von der Worterkennung effizient genutzt werden kann. Es wurden Algorithmen entwickelt, mit denen ein solches Sprachmodell in Form einer kontextfreien Grammatik aus dem ERNEST–Netzwerk des Sprachverarbeitungssystems EVAR extrahiert werden kann. Zu diesem Zweck werden verschiedene Informationen, die in der Wissensbasis enthalten sind, in einen Grammatikformalismus umgesetzt.

ERNEST repräsentiert semantische Netze. Semantische Netze wurden Ende der 60er Jahre als ein grobes Modell des menschlichen Gedächtnisses eingeführt. Informationen über allgemeine Begriffe, Objekte, Ereignisse oder Sachverhalte werden in Knoten und Beziehungen zwischen diesen Begriffen in Kanten repräsentiert. Die Knoten werden in ERNEST durch Konzepte dargestellt. Ein Konzept faßt alle Eigenschaften, die einen linguistischen Begriff betreffen, in Form von Attributen usw. zusammen. Zum Beispiel modelliert das konzept SY_ADV ein Adverb. ERNEST kennt des weiteren die drei Kantentypen Bestandteil, Konkretisierung und Spezialisierung. Die Konzepte sind untereinander durch Kanten von diesen Typen miteinander verbunden. Zulässige Gruppierungen von Bestandteilen und Konkretisierungen werden in den sogenannten Modalitätsbeschreibungen der Konzepte spezifiziert. Durch Kantenrestriktionen, die in den Kantenbeschreibungen festgesetzt sind, kann der Gültigkeitsbereich von Konzepten eingeschränkt werden, auf die die jeweilige Kante verweist.

Aus diesen Informationen wird nun eine kontextfreie Grammatik extrahiert. Aus Aufwandsgründen wurde die Grammatikextraktion auf den Teil der EVAR–Wissensbasis eingeschränkt, der Zeitangaben repräsentiert. Die daraus erzeugte Grammatik stellt also die syntaktische Struktur aller Wortketten dar, die gemäß der Wissensbasis Zeitangaben bilden.

Im Sprachsignal wird dann gezielt nach syntaktisch korrekten Wortfolgen gesucht. Dazu werden Wortfolgen von links nach rechts aufgebaut und, ausgehend von einem bestimmten Startpunkt, mit dem Sprachsignal verglichen. Dieses Signal wird in Segmente unterteilt, wobei jedes Segment als möglicher Anfangspunkt für eine vorgegebene Wortkette dienen kann. Beim Vergleich der erlaubten Wortfolgen mit dem Sprachsignal wird die Ähnlichkeit zwischen beiden bewertet. Gesucht sind schließlich Wortketten, die möglichst gut bewertet sind, also möglichst gut mit dem Sprachsignal übereinstimmen. Aus ihnen werden Wortkettenhypothesen, sogenannte Konstituentenhypothesen, gebildet. Eine solche Hypothese besteht aus einer eindeutigen Identifikation, der Wortkette, dem Anfangs- beziehungsweise Endpunkt im Sprachsignal und einer Bewertung.

Mit Hilfe der aus dem ERNEST–Netz extrahierten Grammatik werden nun Wortfolgen von links

nach rechts aufgebaut. Dazu wird ein aus der Grammatik generierter Parser modifiziert. Ein Parser ist ein Programm, das zu einer gegebenen Folge von Eingabesymbolen berechnet, ob diese Folge in der Sprache enthalten ist, die durch eine zugrundeliegende Grammatik defniert ist. Der hier verwendete Parser wird automatisch durch den Compilergenerator BISON aus der Grammatik erzeugt, die aus der Wissensbasis extrahiert wurde. Während der Generierung von Konstituentenhypothesen wird eine bereits behandelte Wortkette, anfangs die leere Wortkette, um ein Wort erweitert, so daß die neu entstehende Wortkette den Anfang einer korrekten Wortfolge bildet. Es ist also eine Funktion erforderlich, die als Eingabe eine syntaktisch korrekte Wortfolge erhält und als Ergebnis die Menge aller Wörter zurückliefert, mit denen die Eingabefolge gemäß einer Grammatik fortgesetzt werden kann. Man gewinnt diese Funktion durch einige Modifikationen aus dem BISON–Parser.

Die so gewonnenen Wortfolgen werden mit dem Sprachsignal verglichen, das heißt verifiziert, um ein Bewertungsmaß für die Ähnlichkeit zwischen Wortketten und Sprachsignal zu bekommen. Die Wortketten werden mit Hilfe eines statistischen Verfahrens verifiziert, das auf Hidden–Markov–Modellen basiert. Dabei wird zu jeder Wortkette ein Referenzmodell in Form eines Hidden–Markov–Modells erzeugt, das dann mit dem Sprachsignal verglichen wird. Diese Referenzmodelle modellieren die Aussprache der einzelnen Wörter und werden automatisch aus der phonetischen Umschrift der Wörter im Lexikon gewonnen.

Literaturverzeichnis

[AhJo74] A.V. Aho, S.C. Johnson: *LR Parsing*. Computing Surveys, Vol. 6, No. 2, June 1974.

[AhUl72] A.V. Aho, J.D. Ullman: *The Theory of Parsing, Translation and Compiling*, Vol. 1, *Parsing*. Prentice–Hall, Englewood Cliffs, N.J., 1972.

[Diez88] M. Diezel: *Analyse von temporalen Angaben*. Studienarbeit, Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen–Nürnberg, 1988.

[John74] S.C. Johnson: *Yacc: Yet Another Compiler–Compiler*. Bell Laboratories, Murray Hill, New Jersey, 1974.

[Kuhn87] T. Kuhn: *Implementation von Algorithmen zum Vergleich von eindimensionalen Mustern*. Studienarbeit, Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen–Nürnberg, 1987.

[Kuhn89] T. Kuhn: *Eine Suchstrategie zur kontextfreien Analyse von Worthypothesen*. Diplomarbeit, Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen–Nürnberg, 1989.

[Kumm90] F. Kummert: *Flexible Steuerung eines sprachverstehenden Systems mit homogener Wissensbasis*. Dissertation, Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen–Nürnberg, 1990.

[Kunz89] S. Kunzmann: *Die Worterkennung in einem Dialogsystem für kontinuierlich gesprochene Sprache*. Dissertation, Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen–Nürnberg, 1989.

[LeRS83] S.E. Levinson, L.R. Rabiner, M.M. Sondhi: *Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition*. The Bell System Technical Journal, Vol. 62, No. 4, Seiten 1035 – 1074, 1983.

[Manu90] F. Kummert, R. Prechtel, G. Sagerer, S. Schröder: ERNEST–*Manual, Version 1.5*. Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen–Nürnberg, 1990.

[Maye86] O. Mayer: *Syntaxanalyse*. Bibliographisches Institut, Mannheim Wien Zürich, 1986.

[Niem89] H. Niemann: *Pattern Analysis and Understanding*. Springer–Verlag, Belin Heidelberg, 1989.

[Nils71] N.J. Nilsson: *Problem Solving Methods in Artificial Intelligence*. McGraw–Hill, New York, 1971.

[Nöth89] E. Nöth: *Prosodische Information in der automatischen Spracherkennung – Berechnung und Anwendung*. Dissertation, Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen–Nürnberg, 1989.

[Quill68] M.R. Quillian: *Semantic Memory*. In M. Minsky (Editor): *Semantic Information Processing*. MIT Press, Cambridge, MA, Seiten 216 –270, 1968.

[Rabi89] L.R. Rabiner: *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Proceedings of the IEEE, Seiten 257 – 284, 1989.

[Sage90] G. Sagerer: *Automatisches Verstehen gesprochener Sprache*. Bibliographisches Institut, Mannheim, Reihe Informatik, Vol. 74, 1990.