BMBF

Verbmobil

Verbundvorhaben

LR-inkrementelles, probabilistisches Chartparsing von Worthypothesenmengen mit Unifikationsgrammatiken: Eine enge Kopplung von Suche und Analyse.

Hans H. Weber

Univ. Erlangen, IMMD 8

Vm

Report 52
Januar 1995

Januar 1995

Hans H. Weber

Institut für Mathematische Maschinen und Datenverarbeitung 8
Lehrstuhl für Künstliche Intelligenz
Universität Erlangen–Nürnberg
Am Weichselgarten 9
91058 Erlangen

Tel.: (09131) 85 - 9907
Fax: (09131) 85 - 9905
e-mail: `weber@faui80.informatik.uni-erlangen.de`

Gehört zum Antragsabschnitt: 15.7

Die vorliegende Arbeit wurde im Rahmen des Verbundvorhabens Verbmobil vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) unter dem Förderkennzeichen 01 IV 101 G gefördert. Die Verantwortung für den Inhalt dieser Arbeit liegt bei dem Autor.

Wissenschaft ist eine Gruppentätigkeit. Die Gespräche, Diskussionen, Tagungen sind der Boden, auf dem wissenschaftliche Einzelleistungen erst wachsen können. Einen solchen Boden hatte ich während der vergangenen sechs Jahre zunächst bei NATS am Fachbereich Informatik der Universität Hamburg, später dann am IMMD8, dem Lehrstuhl für Künstliche Intelligenz der Universität Erlangen-Nürnberg. Hierfür danke ich den Kollegen beider Institute. Dem Projekt VERBMOBIL und allen seinen Mitarbeitern bin ich ebenfalls zu Dank verpflichtet. Innerhalb des Projekts erhielt ich Gelegenheit, meine Ideen auszuarbeiten und bei zahlreichen Gelegenheiten zur Diskussion zu stellen.

Andreas Hauenstein stellte mir seinen Decoder für Experimente zur Verfügung. Bei der Implementation der probabilistischen Unifikationsgrammatik konnte ich teilweise auf einen von Lutz Euler entwickelten standard Unifikator zurückgreifen. Beiden danke ich ausdrücklich.

Ich danke noch allen Freunden, die mich während des Schreibens an dieser Arbeit ertragen haben. Ohne B. und meine Nachbarin hätte ich es nie geschafft.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Die Erkennung und das Verstehen von menschlicher Sprache durch einen Computer stellt seit Jahren eines der interessantesten Forschungsgebiete der angewandten Informatik dar. Sowohl im Ingenieurbereich, als auch in der empirischen, vor allem an Erkenntniszuwachs interessierten Forschung, ist jedoch noch ein weiter Weg zu gehen, um zu praktikablen Lösungen und umfassenden Theorien zu gelangen.

Informationsdialoge der folgenden Art[1] stellen die maschinelle Sprachverarbeitung bisher vor scheinbar unlösbare Probleme.

Kesseler Espressooo ?

Weber Äh, was .. Espressooo ? Ja klar!

Die Menschen haben die Angewohnheit, nur sehr selten den klaren Normen der sogenannten "Schriftsprache" zu folgen, wenn sie miteinander sprechen. Die kurzen, einfachen – aber vollständigen – Sätze, die von automatischen Spracherkennern erkannt und von automatischen sprachverstehenden Systemen verstanden werden können, sind in der Minderheit.

Die Hoffnung, dem Erkennen und Verstehen der *echten* gesprochenen Sprache näher zu kommen, ist in der jüngeren Vergangenheit verbunden worden mit dem Versuch, die z.T. sehr unterschiedlichen Technologien des Sprachverstehens und der Spracherkennung miteinander zu verbinden. Dieses Unterfangen ist oft als *Speech–Language–Integration* bezeichnet worden.

Die Speech–Language–Integration im Bereich Parsing ist das Thema der vorliegenden Arbeit. Dabei liegt das Hauptinteresse dabei, in beiden Bereichen Standardtechnologie zu verwenden. Zum einen befaßt sich die Arbeit also mit einer Kopplung von Modulen, zum anderen aber auch mit einer Kombination der Methoden. Im Bereich Spracherkennung dominiert die statistische Mustererkennung die Forschungslandschaft. Erkennung von Sprache wird als ein Markovprozess aufgefaßt. Das Sprachverstehen wird in der KI vorwiegend auf der Basis von Constraintsystemen betrieben. Symbolische Sprachbeschreibungen in Form von Unifikationsgrammatiken sind die technische Ausprägung einer Sichtweise des Sprachverstehens als Deduktionsprozess. Hinzu kommt noch die psychologisch basierte Sicht

[1] Typischer Dialog in einem KI-Institut, geführt zwischen Herrn Marcus Kesseler vom IMMD 8 der Friedrich-Alexander-Universität zu Erlangen-Nürnberg, und dem Verfasser. Dieser Dialog kann am Ende der Arbeit **nicht** geparst werden.

des Erkennungs- und Verstehensprozesses als zeitsynchron inkrementellen Vorgang. Diese Auffassung wiederum findet von technischer Seite zunehmend Unterstützung durch Überlegungen zu Parallelisierung integrierter Systeme.

Die vorliegende Arbeit schafft ein Verarbeitungsmodell, das alle drei Sichtweisen vereinbart. Dennoch wird an einigen Stellen der Arbeit auf konservative Argumente wie etwa Komplexitätsbetrachtungen zurückgegriffen. Es zeigt sich, daß dabei hier kein Widerspruch entsteht.

Im ersten Abschnitt der Arbeit (Kap. 2) werden die Standardmodelle und Verfahren vorgestellt, die beim maschinellen Spracherkennen und Sprachverstehen zwischen *Decoding* und *Parsing* derzeit eine prominente Rolle einnehmen. Dabei wird nicht so sehr auf Vollständigkeit geachtet, sondern eher versucht, dem Leser aus angrenzenden Fachgebieten die Grundlage darzulegen, auf der die vorliegende Arbeit aufbaut. Kurze Einführungen werden gegeben bezüglich statistischer Speech- und Languagemodelle, Decoding, Unifikationsformalismen und Unifikationsgrammatiken, Parsing und den klassischen Schnittstellen zwischen Decoding und Parsing. Zu den jeweils knappen Einführungen wird reichlich auf andere einführende und standardisierende Literatur verwiesen.

Der Abschnitt 3 entwickelt einen zeitsynchron inkrementellen interaktiven Aktiven Chartparser (LR-ACP) für Wortlattices. Dabei wird wie folgt vogegangen:

- Wir beginnen mit allgemeinen Komplexitätsbetrachtungen über Latticeparsing in der Chart und bestimmen eine obere Schranke für den kontextfreien Fall.

- Wir definieren eine zeitsynchrone Kontrollschleife für das Parsing von Lattices. Von diesem Ausgangspunkt aus werden die bekannten Techniken für effizientes Parsing im Hinblick auf ihre Anwendbarkeit für LR-ACP diskutiert.

- Eine lose Kopplung von statistischem N-Gramm-Modell und Unifikationsgrammatik wird dargestellt. Die Metrik und Pruningmechanismen sind dabei so generell, daß die in Kapitel 5 entwickelte enge Kopplung ebenfalls einfügbar wird. Die Metrik ist zudem unabhängig anwendbar von Verarbeitungsrichtung oder Parsingstrategie.

- Mit der dargestellten Metrik wird eine Strahlensuche nach rechts und nach links realisierbar. Bei Fehlschlagen der unvollständigen zeitsynchronen Suche kann trotzdem der verbleibende Suchraum noch exploriert werden.

Das anschließende Kapitel 4 beschäftigt sich mit der zeitsynchronen Kopplung des LR-ACP mit einem Beamdecoder in einer zeitsynchron inkrementellen Systemarchitektur.

- Auf der Basis des entwickelten Parsingmodells werden verschiedene zeitsynchrone Protokolle zwischen dem LR-ACP und einem Beamdecoder entwickelt und experimentiert. Zwei verschiedene top down Strategien und eine bottom up Strategie werden verglichen.

- Ein verbreitetes Vorurteil gegenüber zeitsynchronen Verfahren ist, daß die Abbildung von Worthypothesenmengen auf kleine Wortgraphen nicht optimal durchgeführt werden kann. Wir präsentieren ein strikt[2] zeitsynchron inkrementelles Verfahren für diese Abbildung, das eine optimale Zusammenfassung von Gruppen von Worthypothesen leistet.

Das in Abschnitt 3 entwickelte Verarbeitungsmodell ist effizient. Es integriert die Sichtweisen aus Statistik, Deduktion und Psychologie jedoch nur teilweise. Der nächste konsequente Schritt ist die volle Integration von statistischem Languagemodell und Unifikationsgrammatik. Diese Integration wird in Kapitel 5 untersucht. Die Integration wird dabei so vorgenommen, daß die in Abschnitt 4 entwickelten Erweiterungen und Algorithmen für die Interaktion beibehalten werden können.

- Zunächst wird die Palette möglicher Vorgehensweisen für typfreie Unifikationsgrammatiken diskutiert, probabilistische Modelle der Ableitungen zu erstellen. Hierbei wird vor allem auf die Klassenbildung über Merkmalsstrukturen eingegangen.

- Typisierte Unifikationsgrammatiken ermöglichen eine andere Vorgehensweise, die im Detail für die in dieser Arbeit verwendete Grammatik durchexerziert wird. Dabei wird ein doppeltes Modell für Typveränderungen und Regelanwendungen entwickelt und eine Trainingsmethode vorgestellt.

- Die Metrik zur Kantenbewetung aus Abschnitt 3 wird generalisiert auf die Kombination beliebig vieler probabilistischer Modelle.

- Die Arbeit schließt mit einigen Experimenten mit dem LR-ACP und einer probabilistischen typisierten Unifikationsgrammatik, gekoppelt mit einem Beamdecoder.

[2] *Strikt* heißt in diesem Zusammenhang: Ohne Lookahead oder vergleichbare Methoden.

Kapitel 2

Grundlagen der Speech- und Language-Verarbeitung

In dieser Arbeit spielen Teilgebiete der Informatik eine wichtige Rolle, die traditionell nicht immer im Zusammenhang gesehen oder gelehrt werden. Dieses Kapitel gibt daher einen kurzen Überblick bezüglich statistischer Modelle in der Spracherkennung, Grammatiken in der Computerlinguistik und Parsingverfahren in der Sprachverarbeitung, die für Lattice-Parsing geeignet sind. Dabei beschränkt sich der Überblick auf die jeweiligen Bereiche, die für das Thema der Arbeit relevant sind. Standardbegriffe aus dem Compilerbau, wie etwa der CYK-Algorithmus, LR(k)-Grammatiken, usw. werden vorausgesetzt. Hier gibt es gute Lehrbücher, wie etwa Aho et al. [1]. Auch die interne Darstellung von Attribut-Wert-Strukturen oder spezielle Unifikationsalgorithmen können aus Platzgründen nicht eingeführt werden. Einen kurzen aber guten Überblick gibt hier Euler 1991 [28].

2.1 Statistische Modelle

In der klassischen Spracherkennung haben sich die statistischen Verfahren inzwischen durchgesetzt. Die Zuordnung des Schallsignals einer Äußerung zu einer Kette von Wörtern wird reduziert auf ein reines Suchproblem. Die folgende Gleichung stellt dies als Maximierung dar:[1]

$$W_{erkannt} = argmax[P(A|W_i)P(W_i)]$$

Die Erkennung einer gesprochenen Äußerung wird realisiert als Maximierung über dem Produkt der Wahrscheinlichkeit einer Wortkette W_i und der abhängigen Wahrscheinlichkeit einer Folge von Schallereignissen A, gegeben die Wortkette W_i. Hierbei wird $P(A|W_i)$ üblicherweise geliefert von einem Hidden Markov Modell ([75]), während $P(W_i)$ durch statistische Bi-Gramme ([63]) oder Tri-Gramme ([46]) berechnet werden kann.

[1] Die *argmax*-Funktion ist in der Mustererkennung gebräuchlich. Für eine Menge von bewerteten Objekten lieferte *max* den besten Wert zurück, während *argmax* das Objekt zurückliefert, daß den besten Wert erhält.

Man bezeichnet die Menge von HMMs, die in einem System die Abbildung von A nach W_i bewerten auch als *akustisches Modell*, während bei den statistischen Modellen, die die unabhängige Wahrscheinlichkeit einer Wortkette bewerten, von *Sprachmodell* gesprochen wird.

Die akustischen Modelle sind in der vorliegenden Arbeit nicht von zentralem Interesse. In Kapitel 4 wird jedoch stark auf deren Eigenschaften Bezug genommen. Die Schnittstelle zwischen Decoding und Parsing ist in dieser Arbeit zentral. Viele Eigenschaften des hier enwickelten Parsingverfahrens für Worthypothesen sind daher nur im Zusammenhang mit Eigenschaften der Worterkennung vollständig nachvollziehbar. Deshalb enthält der folgende kurze Exkurs über statistische Modelle auch eine kurze Einführung in HMMs und Decoding.

Die statistischen Modelle, die bei der Spracherkennung zum Einsatz kommen, sind die Hidden Markov Modelle, die stochastischen kontextfreien Grammatiken (PCFG) und die stochastischen N-Gramme. Die Hauptmerkmale stochastischer Methoden, welche diese als geeignetes Werkzeug in der Spracherkennung qualifizieren, sind:

- Statistische Modelle sind trainierbar. D.h., es gibt für diese Modelle Methoden, aus großen Mengen von Sprachdaten repräsentative Verteilungen zu ermitteln.

- Statistische Maße ermöglichen eine einheitliche Metrik zwischen verschiedenen Modellen. D.h., aufgrund der Bayes-Regel kann die Konfidenz der Erkennung eines Wortes und die Konfidenz der Erkennung eines Satzes miteinander verrechnet werden.

Die Hidden Markov Modelle, ursprünglich in Arbeiten von Markov 1913 ([60]) und Shannon 1948 und 1951 ([82, 83]) beschrieben, erhielten seit 1970 starken Einfluß auf die Spracherkennung, da mit der Veröffentlichung von Baum, Petrie, Soules und Weiss ([9]) effiziente Verfahren zur Berechnung solcher Modelle möglich wurden. Rabiner und Juang 1986 [75], Rabiner 1988 [74] für die Spracherkennung und allgemein Poritz 1988 [73] sind gute Einführungen in das Thema.

Statistische kontextfreie Grammatiken werden zuerst bei Baker 1979 [8] behandelt. Dort wird eine Variante des Vorwärts–Rückwärts–Verfahrens für SCFGs entwickelt, der Inside–Outside–Algorithmus. Effizientes Parsing wird bei Fujisaki 1984 [32] präsentiert. Sehr klar und einfach werden die SCFGs in Fujisaki, Jelinek, Cocke, Black und Nishino 1991 [33] vorgestellt. Eine weitere Einführung ist Jelinek 1992 [44]. Anwendungen im Speech-Bereich sind Corazza, Gretter, Satta und De Mori 1991 [19], Wrigley und Wright 1991 [99].

N-Gramme, die einfachsten Vertreter statistischer Modelle, werden u.a. bei Nadas 1985 [63], Katz 1987 [49] und Kesselheim & Bloemberg 1988 [52] behandelt.

2.1.1 Hidden Markov Modelle

Ein HMM ist ein endlicher schreibender Automat, dessen Übergänge zwischen Zuständen stochastisch gewichtet sind, und bei dem für jeden Zustand stochastisch gewichtet ist, welches Symbol[2] geschrieben (emittiert) wird. Ein HMM läßt sich formal beschreiben als:

[2] Der Einfachheit halber betrachten wir hier nur HMMs mit diskreten Emissionswahrscheinlichkeiten.

Definition 2.1 *Ein HMM λ ist*

S $= \{S_1, S_2...., S_N\}$, *die individuellen Zustände von λ. Wir nennen $q_t \in \mathbf{S}$ den aktuellen Zustand zum Zeitpunkt t.*

V $= \{V_1, V_2...., V_M\}$, *das Symbolalphabet von λ.*

O $= O_1, O_2....., O_T$ *die aktuelle Beobachtungssequenz von Symbolen aus V zu den Zeitpunkten t, $1 \leq t \leq T$.*

B $= \{b_j(k) | b_j(k) = P[V_k = O_t | q_t = S_j]\}$.

B ist also eine Menge von Wahrscheinlichkeiten, jeweils für Paare von Zuständen und Symbolen, daß das zu einem Zeitpunkt t beobachtete Symbol O_t eben das Symbol V_k ist, falls der aktuelle Zustand zu diesem Zeitpunkt, q_t, der Zustand S_j ist.

Wir nehmen der Einfachheit halber an, daß die Werte in B für alle Zustände und Symbole größer als 0 sind.

Es gilt immer $\sum_{k=1}^{M} b_j(k) = 1$, für jeden Zustand $S_j \in \mathbf{S}$.

A $= \{a_{ij} | a_{ij} = P[q_{t+1} = S_j | q_t = S_i]\}$, *wobei $1 \leq i,j \leq N$.*

In A ist für jedes Paar von Zuständen die Übergangswahrscheinlichkeit angegeben, mit der von einem in den anderen Zustand gewechselt werden kann.

Es gilt immer $\sum_{j=1}^{N} a_{ij} = 1$.

Π $= \{\pi_j | \pi_j = P[q_1 = S_j]\}$, $1 \leq j \leq N$.

Für jeden Zustand ist eine Anfangswahrscheinlichkeit definiert, wiederum mit $\sum_{j=1}^{N} \pi_j = 1$.

Kurz kann ein HMM durch die drei wichtigsten Parameter $\lambda = \{A, B, \Pi\}$ beschrieben werden.

Die wichtigsten Algorithmen bezüglich HMMs betreffen:

1. Die Ermittlung einer Bewertung für eine Beobachtungssequenz mit gegebenem Modell, $P[O|\lambda]$.

   Bei der Erkennung von Wörtern, Phonen oder anderen Einheiten wird über den konkurrierenden Modellen für jede dieser Einheiten maximiert. Das beste Modell gehört zur erkannten Einheit.

2. Die rekursive Bestimmung der Bewertung für einzelne Zustände $P[q_t = S_j | O, \lambda]$ und damit des besten Pfades durch ein Modell, $S_1...S_T$, gegeben O, λ.

3. Die Ermittlung der Parameter des Modells λ aus einer Beobachtungssequenz.

2.1.1.1 Vorwärts–Rückwärts–Bewertung

Die Suche nach $P[O|\lambda]$ ist das Problem, wie die Wahrscheinlichkeit für eine bestimmte Beobachtungssequenz $O = O_1..O_T$ aus λ errechnet werden kann. Die Zustände von λ können, während O gelesen bzw. erzeugt wird, in jeder beliebigen Reihenfolge durchlaufen werden, jeweils mit einer bestimmten Wahrscheinlichkeit. Bei N Zuständen gibt es also N^T verschiedene Folgen $Q = q_1..q_T$ von Zustandspfaden.

Die Wahrscheinlichkeit, daß ein Pfad Q die Sequenz O erzeugt, ergibt sich aus den Emissionswahrscheinlichkeiten für jedes Paar q_t, O_t, also den dazugehörigen $b_{q_t}(O_t)$, und den Übergangswahrscheinlichkeiten des Pfades. Der erste Schritt auf dem Pfad wird natürlich nicht durch A, sondern durch Π gegeben:

$$P[Q, O|\lambda] = \pi_{q_1} b_{q_1}(O_1) * \prod_{t=2}^{T} a_{ij} b_{q_t}(O_t) \text{ ,für } q_t = S_j, q_{t-1} = S_i \tag{2.1}$$

Die Summe der Wahrscheinlichkeiten aller Pfade ergibt schließlich die gesuchte Wahrscheinlichkeit, mit der λ O erzeugt bzw. liest.

$$P[O|\lambda] = \sum_{Q} P[Q, O|\lambda] \tag{2.2}$$

Es ist sehr ineffizient, alle Pfade aufzuzählen, die entsprechenden Wahrscheinlichkeiten auszurechnen und dann erst aufzusummieren. Das Vorwärts-Rückwärts-Verfahren nutzt aus, daß sich viele verschiedene Pfade nur in wenigen Zuständen unterscheiden.

Falls die Beobachtungssequenz nur die Länge 1 hat, dann gibt es N verschiedene Pfade Q: Alle Zustände in S können $Q = q_1$ sein. Falls wir die Beobachtungssequenz um 1 verlängern auf die Länge 2, dann gibt es N^2 verschiedene Pfade. Die erste Hälfte aller Pfade ist jedoch bereits errechnet. Es müssen daher lediglich von allen möglichen Pfaden der Länge 1 die Verlängerungen ausgerechnet werden. Jede Verlängerung um 1 umfaßt N^2 Schritte (Rechnungen) und die gesamte Kalkulation kostet N^2T.

Die Iteration für $O = O_1...O_T$ kann von 1 nach rechts, oder von T nach links durchgerechnet werden. Man nennt dies Vorwärts– bzw. Rückwärts–Kalkulation:

Definition 2.2 *Iteration über die Vorwärtsvariable α :*

$$\alpha_1(i) = \pi_i b_i(O_1) \quad 1 \le i \le N \tag{2.3}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \quad \begin{array}{l} 1 \le t \le T-1 \\ 1 \le j \le N \end{array} \tag{2.4}$$

$$P[O|\lambda] = \sum_{i=1}^{N} \alpha_T(i) \tag{2.5}$$

Definition 2.3 *Iteration über die Rückwärtsvariable β :*

$$\beta_T(i) = 1 \quad 1 \le i \le N \tag{2.6}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}\beta_{t+1}(j) \quad \begin{array}{l} t = T-1, T-2 ... 1 \\ 1 \leq i \leq N \end{array} \tag{2.7}$$

$$P[O|\lambda] = \sum_{i=1}^{N} \pi_i b_i(O_1)\beta_1(i) \tag{2.8}$$

2.1.1.2 Zustandsfolgen

Diese zwei Möglichkeiten, die Pfade in einem HMM vorwärts oder rückwärts durchzurechnen, sind interessant, da man von jedem Beobachtungszeitpunkt t aus $P[O|\lambda]$ errechnen kann, mit:

$$P[O|\lambda] = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) \tag{2.9}$$

Die obigen Summanden sind alle Pfade von 1 bis t, die in q_i münden (der α-Teil) und von t bis T weitergehen, von q_i aus (der β-Teil). Aufsummiert über alle möglichen q_i ergibt dies wieder alle Pfade.

Damit ist auch eine Lösung für Problem 2 gegeben: Die relative Wahrscheinlichkeit für einen Zustand zu einem bestimmten Zeitpunkt ist die absolute Wahrscheinlichkeit, geteilt durch die Summe der Wahrscheinlichkeiten für alle Zustände:

$$P[S_i = q_t|O] = \gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P[O|\lambda]} \tag{2.10}$$

Die Folge bester Zustände und die beste Folge von Zuständen in einem HMM sind nicht identisch (obwohl es der Fall sein kann). Die Folge bester Zustände wird ermittelt, indem mit obiger Gleichung $\gamma_t(i)$ über i maximiert wird, sukzessive für alle t von 1 bis T. Die beste Folge von Zuständen wird durch den Viterbi-Algorithmus ([31]) ermittelt. Hierbei wird iterativ vorgegangen wie bei der Vorwärts-Methode. Bei jedem Schritt von t nach $t+1$ wird nicht wie oben die Summe gebildet, sondern nur der beste Pfad bis zu jedem möglichen neuen Zustand expandiert. Dabei wird ausgenutzt, daß, wenn wir bereits die besten Wege bis zu jedem Zustand aus S zum Zeitpunkt t kennen, dann kann jeder beste Weg bis zu einem Zustand aus S zum Zeitpunkt $t+1$ nur die Verlängerung eines dieser Wege sein.

Definition 2.4 *VITERBI-Suche des besten Pfades durch ein HMM*

$$\begin{array}{l} \delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \\ \phi_1(i) = 0 \end{array} \tag{2.11}$$

$$\begin{array}{ll} \delta_t(j) = max_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(O_t), & 2 \leq t \leq T, 1 \leq j \leq N \\ \phi_t(j) = argmax_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right], & 2 \leq t \leq T, 1 \leq j \leq N \end{array} \tag{2.12}$$

$$\begin{array}{l} P^{max} = max_{1 \leq i \leq N} \left[\delta_T(i)\right] \\ q_T^{max} = argmax_{1 \leq i \leq N} \left[\delta_T(i)\right] \end{array} \tag{2.13}$$

$$q_t^{max} = \phi_{t+1}(q_{t+1}^{max}), \quad t = T-1, T-2, ..., 1 \tag{2.14}$$

Eine Implementation des VITERBI–Algorithmus kann mit Hilfe von δ in 2.11 und 2.12 die Pfade vorwärts durchrechnen und sich die Ergebnisse merken, bis zum Abbruch für δ in 2.13. Das Ergebnis P^{max} ist die Bewertung des besten Pfades durch das HMM. Mit dem rekursiven Abstieg wie in 2.14 kann schließlich der Pfad selbst errechnet werden.

2.1.1.3 Baum–Welch–Lernen

Es gibt kein analytisches Verfahren, um die Parameter A, B und π eines HMM zu ermitteln. Der Baum-Welch-Algorithmus ist ein Näherungsverfahren, bei dem mit einem HMM mit zufällig gewählten Parametern eine Stichprobe O beobachtet und bewertet wird. Aufgrund der Beobachtung O und dem HMM λ werden für alle Zustandsübergänge und Emissionen Wahrscheinlichkeiten errechnet. Diese bilden dann die Parameter eines neuen Modells $\overline{\lambda}$. Mit dem neuen Modell wird diese Prozedur wiederholt, usw..

Das Abbruchkriterium ist gegeben, wenn die Erkennungsrate nach einem Schritt nicht mehr steigt, d.h., daß die Modellwahrscheinlichkeit für die Beobachtungssequenz selbst – die Trainingsstichprobe – nicht mehr größer wird.

In 2.10 wurde angegeben, wie die Wahrscheinlichkeit errechnet werden kann, zum Zeitpunkt t im Zustand S_i zu sein, gegeben O und λ. Mit der gleichen Methode können wir angeben, wie wahrscheinlich es ist, zum Zeitpunkt t in S_i zu sein und zum Zeitpunkt $t+1$ in S_j:

$$P\left[S_i = q_t, S_j = q_{t+1}|O\right] = \xi_t(ij) = \frac{\alpha_t(i)a_{ij}b_j(O_t+1)\beta_{t+1}(j)}{P\left[O|\lambda\right]} \tag{2.15}$$

Wenn wir $\gamma_t(i)$ und $\xi_t(ij)$ jeweils aufsummieren, über alle Beobachtungszeitpunkte der Stichprobe bzw. Beobachtungssequenz O, dann erhalten wir Werte, die darstellen, wie oft Zustand S_i insgesamt durchlaufen wird, bzw., wie oft von S_i nach S_j gewechselt wird. [3]

$$\textstyle\sum_{t=1}^{T-1} \gamma_t(i) = \quad \text{Erwartete Häufigkeit Übergänge von} \quad S_i \tag{2.16}$$

$$\textstyle\sum_{t=1}^{T-1} \xi_t(ij) = \quad \text{Erwartete Häufigkeit Übergänge} \quad S_i \quad S_j \tag{2.17}$$

Das neue Modell $\overline{\lambda}$ wird nun wie folgt ermittelt. $\overline{\pi_i}$ für jeden Zustand S_i ist einfach die Wahrscheinlichkeit in Zustand S_i zu sein, zum Zeitpunkt $t = 1$:

$$\overline{\pi_i} = \gamma_1(i) \tag{2.18}$$

Die neuen Übergangswahrscheinlichkeiten a_{ij} sind jeweils die erwarteten Übergänge von S_i nach S_j geteilt durch die Erwartung in S_i zu sein, bzw. von dort zu irgendeinem Folgezustand zu wechseln (also normalisiert) :

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(ij)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.19}$$

[3] Dem aufmerksamen Leser wird auffallen, daß in 2.16 und in 2.17 nur bis $T-1$ aufsummiert ist. Der Grund ist, daß ja zum Zeitpunkt T kein Übergang mehr stattfindet. Wenn wir die Übergänge berechnen wollen, gehört T nicht mehr mit in die Summe hinein. Wenn wir die Aufenthalte berechnen wollen, wie in 2.20 jedoch schon.

Sehr ähnlich ist die Schätzung der Emissionswahrscheinlichkeiten des neuen Modells. Die Anzahl der erwarteten Aufenthalte in S_j verteilt sich auf die Beobachtung der verschiedenen Symbole aus V. $\overline{b_j(k)}$ muß also mit dem Anteil aller Beobachtungen aus S_j korrelieren, bei dem V_k beobachtet wird. Es ergibt sich aus der erwarteten Anzahl der Aufenthalte in S_j, bei denen V_k beobachtet wird, geteilt durch die Anzahl aller Aufenthalte in S_j (normalisiert):

$$\overline{b_j(k)} = \frac{\sum_{t=1, O_t=V_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{2.20}$$

2.1.2 Probabilistische Kontextfreie Grammatiken (PCFG)

Eine probabilistische kontextfreie Grammatik ist eine kontextfreie Grammatik, deren Produktionen mit Wahrscheinlichkeiten versehen sind. Dies ermöglicht es, für eine von einer PCFG erzeugte Wortkette zu bestimmen, mit welcher Wahrscheinlichkeit die Wortkette von der PCFG erzeugt wird und welche Wahrscheinlichkeit eine bestimmte Ableitung der PCFG für die Eingabe hat.

Definition 2.5 *Eine PCFG G ist:*

N $= \{n_1..n_n\}$, *die Menge nichtterminaler Symbole.*

T $= \{t_1..t_m\}$, *die Menge terminaler Symbole.*

V $= N \cup T$. *Das sind alle Symbole der Grammatik.*

S , *das Startsymbol der Grammatik,* $S \in N$.

R $\subseteq N \times V^*$, *die kontextfreien Produktionen von G.*

P $= \{P(r_i) | r_i = \alpha \rightarrow \beta \ , \ \sum_\gamma P(\gamma|\alpha) = 1\}$.

Für jede Produktion in R ist in P eine Wahrscheinlichkeit dafür definiert, daß der rechte Teil der Regel erzeugt wird, gegeben den linken Teil der Regel.

2.1.2.1 Die Wahrscheinlichkeit einer Wortkette bei PCFGs

Eine CFG ist ein Erzeugungsmodell für Wortketten über einem Alphabet von Wörtern. Üblicherweise wird die Sprache einer CFG definiert über eine Ableitungsrelation $\Rightarrow$ und deren transitiven Abschluß $\stackrel{*}{\Rightarrow}$:

Definition 2.6 $\Rightarrow$ *und* $\stackrel{*}{\Rightarrow}$

$A\alpha C \Rightarrow A\beta C$, *falls* $\alpha \rightarrow \beta \in R$.

$X \stackrel{*}{\Rightarrow} Y$, *falls* $X \Rightarrow Y$

$X \stackrel{*}{\Rightarrow} Z$, *falls* $X \stackrel{*}{\Rightarrow} Y$ *und* $Y \Rightarrow Z$

Die Sprache einer Grammatik ist definiert über:

$$\{W | W \in T^*, S \overset{*}{\Rightarrow} W\} \tag{2.21}$$

Es kann mehrere Ableitungen geben für eine Wortkette W. Wir bezeichnen diese Ableitungen als $D(W) = \{D_1(W)..D_n(W)\}$. In einer PCFG kann für jede Ableitung eine Wahrscheinlichkeit errechnet werden:

$$P(A\alpha B \Rightarrow A\beta B) = P(\beta|\alpha) \tag{2.22}$$

und rekursiv:

$$P(S \overset{*}{\Rightarrow} W) = P(S \Rightarrow X) * P(X \overset{*}{\Rightarrow} W) \tag{2.23}$$

Das Produkt der Wahrscheinlichkeiten aller Einzelregeln bildet die Wahrscheinlichkeit einer Ableitung. Die Wahrscheinlichkeit, mit der eine Wortkette erzeugt wird, ist die Summe der Wahrscheinlichkeiten aller Ableitungen:

$$P(W|G) = \sum_j P(D_j(W)) \tag{2.24}$$

Es gibt mehrere Algorithmen, die Wahrscheinlichkeiten einzelner Ableitungen zu berechnen, während diese Ableitungen aufgebaut werden. Sie fußen im wesentlichen auf den klassischen Parsingverfahren des Compilerbaus und werden direkt im Speechparsing angewendet.

2.1.2.2 Schätzen der Parameter einer PCFG

Grundsätzlich sind drei Verfahren für die Schätzung der Parameter einer PCFG bekannt. Das einfachste Verfahren besteht aus einfachem Abzählen und Normalisieren der Regelanwendungen, wobei für jeden Satz der Stichprobe aus allen möglichen Ableitungen jeweils die erwünschten von den falschen getrennt werden müssen. Dieses kann nur von menschlichen Experten geleistet werden und stellt in der Praxis ein großes Problem dar.[4]

Die beiden anderen Methoden sind Näherungsverfahren, die dem Baum-Welch-Verfahren für HMMs entlehnt sind. Hier wird eine von uns überarbeitete Version des einfacheren Verfahrens aus Fujisaki et al. 1991 ([33]) kurz dargestellt. Komplizierter, aber wesentlich effizienter ist der Inside-Outside-Algorithmus ([8], [44]), der auf dem CYK-Parsingverfahren und dem Baum-Welch-Algorithmus beruht.

Das folgende Verfahren geht davon aus, daß eine bestimmte grammatische Ableitung, mit der eine beobachtete Wortfolge erzeugt wird, "hidden" ist, wie eine bestimmte Zustandsfolge eines HMMs. Durch die Schätzung der Parameter wird also nicht nur ermittelt, wie wahrscheinlich die Emmission einer Wortfolge ist, sondern auch, wie wahrscheinlich einzelne Ableitungen dieser Wortfolge sind. In Experimenten konnte von Fujisaki et al. ([33]) gezeigt werden, daß die somit sich ergebenden Präferenzen bestimmter Lesarten mit der linguistischen Intuition weitgehend übereinstimmen.

Definition 2.7 *Trainingsprozedur für PCFGs:*

[4] Sogenannte *Baumbanken*, mit vorgegebenen Ableitungsbäumen für gegebenen Äußerungen sind mühevoll von Hand zu erstellen.

1. *Initialisiere die Wahrscheinlichkeiten* $P(\beta|\alpha)$*, sodaß* $\sum_\beta P(beta|\alpha) = 1$

2. *Für alle Sätze* $B^i \in \left[B^1..B^n\right]$ *der Stichprobe, finde alle Ableitungen* D^i_j

3. *Berechne deren Wahrscheinlichkeiten als* $P(D^i_j) = \prod_{r\in D^i_j} P(r)$*, das Produkt der Wahrscheinlichkeiten aller Regelanwendungen.*

4. *Berechne einen nach relativen Wahrscheinlichkeiten der Ableitungen gewichteten Zähler* $C^i_\alpha(\beta)$ *für die Regelanwendungen von* $\alpha \rightarrow \beta$*, wobei* $count^i_j(\alpha, \beta)$ *die absolute Anzahl der Regelanwendungen* $\alpha \rightarrow \beta$ *in Ableitung* D^i_j *ist.*

$$C^i_\alpha(\beta) = \sum_j \left(\frac{P(D^i_j)}{\sum_k P(D^i_k)} count^i_j(\alpha, \beta) \right) \tag{2.25}$$

5. *Normalisiere den Zähler zuerst über alle Regeln der selben Variablen und dann über alle Sätze der Stichprobe, in der die Variable in einem Pars vorkam. Beachte immer nur solche Sätze der Stichprobe für eine Variable* α*, für die ein* D^i_j *existiert, sodaß* α *in* D^i_j *vorkommt.*[5]

$$f_\alpha(\beta) = \frac{\sum_{i,\alpha\in D^i_j} \frac{C^i_\alpha(\beta)}{\sum_\gamma C^i_\alpha(\gamma)}}{\sum_{i,\alpha\in D^i_j} 1} \tag{2.26}$$

6. *Ersetze die Parameter* $P(\beta|\alpha)$ *durch* $f_\alpha(\beta)$*. Gehe wieder zu Schritt 3 und brich ab, wenn in einer Iteration keine Veränderung der Parameter mehr stattfindet.*

Das Verfahren von Fujisaki et al. 1991 [33] ist hier etwas modifiziert dargestellt (Siehe Fußnote 5). In Abschnitt 5 verwenden wir es in der hier präsentierten Form, um die Ableitungen einer Unifikationsgrammatik zu trainieren.

2.1.3 N-Gramm-Modelle

N-Gramme sind die einfachsten Vertreter statistischer Modelle. Ein N-Gramm besteht aus einer Liste von Wahrscheinlichkeiten für Wörter, gegeben den linken Kontext von Wörtern der Länge $N-1$. Gebräuchlich sind Bi- und Tri-Gramme — also Wortpaare und Worttripel. Aus kombinatorischen Gründen ist es nicht möglich, die Wahrscheinlichkeiten für längere Wortketten direkt aus Korpora zu extrahieren. Daher berechnet man die Wahrscheinlichkeit einer Kette von Wörtern mit beliebiger Länge als Aproximation aus verrechneten Bi-Gramm-Wahrscheinlichkeiten bzw. Tri-Gramm-Wahrscheinlichkeiten:

Definition 2.8 *Bi-Gramm-Approximation*

$$P(W_n|W_1, W_2, ...W_{n-1}) \approx \prod_{i=1}^{n} P(W_i|W_{i-1}) \tag{2.27}$$

[5] In Fujisaki et al. ([33]), Seite 143, ist das Verfahren stark vereinfacht dargestellt. Wenn es wörtlich wie in [33] implementiert wird, gerät in Gleichung 2.26 eine 0 in einen Nenner. Es fehlt dort auch die Normalisierung über alle Sätze der Stichprobe.

Definition 2.9 *Tri-Gramm-Approximation*

$$P(W_n|W_1, W_2, ...W_{n-1}) \approx \prod_{i=1}^{n} P(W_i|W_{i-1}, W_{i-2}) \tag{2.28}$$

Hierbei sind W_0 und W_{-1} die linke Grenzmarkierung der Wortkette. Das Schätzen der Parameter für Bi-Gramme und Tri-Gramme erfolgt direkt über das Abzählen und Normalisieren der in einer Stichprobe beobachteten Vorkommnisse der Paare und Tripel.

Definition 2.10 *Schätzen der Bi-Gramm-Wahrscheinlichkeiten*

$$P(W_i|W_j) = \frac{count(W_j, W_i)}{count(W_j)} \tag{2.29}$$

Definition 2.11 *Schätzen der Tri-Gramm-Wahrscheinlichkeiten*

$$P(W_i|W_j, W_k) = \frac{count(W_j, W_k, W_i)}{count(W_j, W_k)} \tag{2.30}$$

Mit zunehmendem N eines N-Gramms steigt die benötigte Stichprobe exponentiell. Die Approximation wird dafür exakter. Da niemals auszuschließen ist, daß Wortkombinationen möglich sind, die in der Stichprobe nicht beobachtet wurden, werden alle nicht beobachteten Kombinationen mit einer Wahrscheinlichkeit initialisiert, die kleiner ist als die kleinste Wahrscheinlichkeit einer beobachteten Kombination. Die ermittelten Werte werden dann soweit verändert, daß die Summe der Wahrscheinlichkeiten aller Kontexte für ein Wort wieder 1 ergibt. Das intelligente *Smoothing* von N-Gramm-Modellen ist inzwischen ein ganzer Forschungsbereich geworden. Hier wird nicht weiter darauf eingegangen.

Da der benötigte Speicher für Bi-Gramme $O(n^2)$ und für Tri-Gramme $O(n^3)$ ist, sind in der Praxis die Wörter des N-Gramms subterminale Symbole und nicht natürlichsprachliche Wörter.

Die Klassenzuordnung von Wörtern zu subterminalen Symbolen läßt sich teilweise automatisch durchführen. Kneser und Ney 1991 [54] schlagen ein heuristisches Verfahren vor, bei dem die Wahrscheinlichkeit der Erzeugung einer Stichprobe über verschiedenen Klassenannahmen maximiert wird und erzielen gute Ergebnisse.

2.1.4 Decoding

Ein Decoder ist ein Programm, das für eine gegebene Darstellung eines Sprachsignals in Frames[6] erkannte Einheiten ausgibt. Die Einheiten sind häufig Äußerungs- oder Wortgroß, aber es sind auch andere Granulate üblich. Der Beam Decoder, der in Abschnitt 4 mit dem LR-ACP gekoppelt wird, ist ein Worthypothesenerkenner, d.h. die Modelle die mit den Signalabschnitten verglichen werden, stellen Wortformen dar.

Es lassen sich Beam und Stack Decoder unterscheiden. Beamdecoder verfolgen alle Pfade von links nach rechts parallel und verwenden eine Viterbi-Suchstrategie. Dabei kann mit oder ohne den Rückwärtsschritt gearbeitet werden.

[6] Frames sind Segmente, meist mit zeitlich gleicher Ausdehnung von 10ms. Es sind jedoch auch andere Formate verbreitet.

Der in dieser Arbeit verwendete Decoder arbeitet ohne Rückwärtsschritt. Der Rückwärtsschritt wird vom LR-ACP mit übernommen.

Stack Decoder verwenden eine A*-Suche, wobei die lokalen Vergleiche mit einer Viterbi- oder Vorwärts-Rückwärts-Suche durchgeführt werden können.

Aktuelle Arbeiten zu Beam Decoding sind Ney et al. 1992 [64], Ney 1993 [66] und Aubert et al. 1994 [6]. Stack Decoder werden unter anderem in Chow & Schwartz 1989 [18] oder Schwartz & Austin 1990 [80] entwickelt.

Der Beam Decoder der in Abschnitt 4 mit dem LR-ACP zeitsynchron gekoppelt wird, wurde von Andreas Hauenstein an der Universität Hamburg entwickelt. Die Besonderheit dieses Decoders ist, daß links rechts inkrementell erkannte Wortformen sofort als Hypothesen ausgegeben werden, sobald ein Endezustand des korrespondierenden HMMs oberhalb des Suchstrahls liegt. Der Decoder arbeitet ohne Wortkopien, d.h., wenn ein Modell aktiv ist, kann es nicht erneut gestartet werden.

2.2 Unifikationsgrammatiken

In Sprachverstehenden Systemen werden natürlichsprachlichen Äußerungen, die als Wortfolge vorliegen, grammatische Ableitungsstrukturen zugeordnet, welche die kompositionale Bedeutung der Eingabe entweder explizit als Ergebnis der Ableitung darstellen oder aber implizit aus der ganzen Ableitungsfolge rekonstruierbar machen. Die Ableitungen selbst werden durch sukzessive Anwendungen grammatischer Operationen gebildet. Die Unifikationsgrammatiken haben sich hierbei inzwischen als Hauptwerkzeug der Repräsentation grammatischen Wissens etabliert.

Die grundsätzliche Vorgehensweise bei der Zuordnung einer logischen Form zu einer Eingabe in natürlicher Sprache ist dabei:

1. Ordne jedem Wort der Eingabe die passenden grammatischen Beschreibungen in Form von Ausdrücken der Unifikationsgrammatik aus einem Lexikon zu.
2. Wende die Operationen der Unifikationsgrammatik an, bis eine grammatische Beschreibung der ganzen Wortfolge entsteht.

Eine auf diese Weise ermittelte Beschreibung enthält Information über die grammatischen Eigenschaften der Eingabe, meistens syntaktische und semantische Information, in manchen Ansätzen auch pragmatische Information.

Die Unifikation als zentrale grammatische Operation wurde zuerst von Kay 1979 vorgeschlagen ([50],[51]). Es gibt eine große Zahl verschiedener Ansätze von Unifikationsgrammatiken, wobei einige direkt der theoretischen Linguistik entstammen, andere hingegen als spezielle Werkzeuge der maschinellen Sprachverarbeitung entwickelt wurden. Zur ersten Gruppe zählen u.a. LFG[7], HPSG[8], GPSG[9] und CUG[10]. Typische Varianten von Unifikationsgrammatiken, die aus der Informatik stammen, sind UTAG[11] oder DCG[12]

---

[7] Lexical Functional Grammar[10].
[8] Head-Driven Phrase Structure Grammar [72].
[9] Generalized Phrase Structure Grammar [34].
[10] Categorial Unification Grammar [94],[101])
[11] Unification Tree Adjoining Grammars.
[12] Definite Clause Grammars[70].

Ein Grammatiktyp für die Analyse von natürlicher Sprache besteht grundsätzlich aus einem Formalismus, einem Ableitungsschema und einem Lexikalischen Schema. Darüber hinaus ist natürlich ein gewisser Stil des Grammatikschreibens von Bedeutung, den wir aber an dieser Stelle vernachlässigen wollen. Während bei kontextfreien Grammatiken der Formalismus so einfach ist, daß er üblicherweise nie spezifiziert wird, hat bei Unifikationsgrammatiken der Formalismus eine tragende Bedeutung.

Ein Grammatikformalismus definiert Datenstrukturen und Operationen über diesen.

Bei einer kontextfreien Grammatik besteht der Formalismus aus einer Datenstruktur und einer Operation. Die Datenstruktur ist die atomare Variable, die Operation ist der Test auf Identität.

Die Datenstrukturen und Operation von Unifikationsformalismen werden in diesem Abschnitt informell beschrieben.

Das Ableitungsschema stellt einen Zusammenhang her zwischen den grammatischen Operationen und einer grammatischen Ableitungsrelation $\Rightarrow$.

Bei PATR-artigen Unifikationsgrammatiken gilt ein ähnliches Ableitungsschema wie bei Phrasenstrukturgrammatiken. Grammatikregeln sind hier Ersetzungsregeln. Beim Ersetzen eines grammatischen Objekts durch eine linke Regelseite wird das Ergebnis jedoch durch die Unifikation der beiden erzeugt, anstatt daß nur ein Identitätstest durchgeführt wird, wie bei den Phrasenstrukturgrammatiken. Dadurch sind viele Techniken für kontextfreie Grammatiken auch für PATR-artige Unifikationsgrammatiken anwendbar. Die in dieser Arbeit verwendete Unifikationsgrammatik ist eine PATR-artige Gramamtik.

2.2.1 Unifikationsformalismen

Die Bereiche der Informatik, in der Unifikation eine wichtige Rolle spielt, sind das Theorembeweisen, die Logikprogrammierung und das automatische Sprachverstehen.

Die Wurzeln liegen in Arbeiten zur Unifikation von Termen erster Ordnung von Herbrand 1971 [42] und Robinson 1965 [76].

Die Unifikation von Termen erster Ordung (mit fester Arität), die aus Funktionssymbolen (f,g), Variablen (x,y) und Konstanten (a) aufgebaut sind, wird als das Problem formuliert, für zwei Terme eine Substitution von Variablen durch Terme zu finden, sodaß beide Terme gleich werden.

So sind zum Beilspiel die Terme f(x,y) und f(g(y,a),h(a)) unifizierbar, indem y durch h(a) ersetzt wird und x durch g(h(a),a). Das Ergebnis der Termunifikation ist dann f(g(h(a),a),h(a)).[13]

Effiziente Algorithmen zur Termunifikation verwenden strukturteilende Graphendarstellungen von Termen. Veröffentlichungen in dieser Richtung sind Huet 1976 [43], Ait-Kaci 1984 [2] oder Godden 1990 [36].

Unifikationsformalismen im Bereich Sprachverstehen verwenden als Terme Merkmalsstrukturen (s.u.). Die Semantik solcher Formalismen als Logik ist u.a. in Rounds & Kasper 1986 [77], Moshier & Rounds 1987 [61], Johnson 1987 [47] oder Smolka 1988 [88] behandelt worden. Wir gehen in dieser Arbeit nicht hierauf ein. Operationen über Merkmalsbeschreibungen werden nur syntaktisch behandelt.

[13] Das Beispiel ist Shapiro 1992 [84] entnommen.

$$\begin{bmatrix} \text{Feature1:} & \begin{bmatrix} \text{Feature2:} & \text{Value1} \end{bmatrix} \\ \text{Feature3:} & \text{Value2} \end{bmatrix}$$

Abbildung 2.1: Einfache Merkmalsstruktur.

$$\begin{array}{lcl} \text{Feature1:Feature2} & = & \text{Value1} \\ \text{Feature3} & = & \text{Value2} \\ \text{Feature4} & = & \text{Feature5} \end{array}$$

Abbildung 2.2: Eine Merkmalsstruktur als Menge von Pfadgleichungen mit einer Koreferenz.

Einführungen in Unifikationsformalismen und ihre Verwendung zur Sprachbeschreibung finden sich in Shieber 1986 [87], Pereira 1987 [69] und Pollard & Sag 1987 [72].

2.2.1.1 Merkmale und Werte

Zur Beschreibung von sprachlichen Zeichen wurden mit Kay 1979 [50] eine spezielle Art von Termen eingeführt, die Merkmalsstrukturen.

Eingebettete Substrukturen von Merkmalsstrukturen werden durch Symbolnamen (Merkmale) benannt und nicht mehr durch die Stelle im Term. Eine feste Arität ist ebenfalls nicht gefordert.

Ein Beispiel für eine Merkmalsstruktur ist Abbildung 2.1.

Merkmale (Features) können rekursiv eingebettet sein. Am Ende der Einbettung steht etweder eine Variable oder ein Atom als Wert (Value).

2.2.1.2 Koreferenz

Merkmalsstrukturen lassen sich auch als Mengen von Gleichungen darstellen. Die Merkmalsstruktur aus Beispiel 2.1 wäre als Gleichungen geschrieben wie in Abbildung 2.2

Atomare Werte werden bei der Darstellung von MS als Gleichungen mit der Konkatenation der Merkmale – einem Pfad – gleichgesetzt. Indem z.B. zwei Pfade direkt gleichgesetzt werden, wird ausgedrückt, daß zwei Merkmale den selben Wert teilen. In der MS wird diese Koreferenz durch gleiche Variablen dargestellt, die gebunden oder ungebunden sein können.

Merkmalsstrukturen werden oft als gerichteter azyklischer Wurzelgraph dargestellt. Die Darstellung als Pfadgleichungen listet alle in einem solchen Graphen enthaltenen Pfade auf, wobei eine Strukturteilung im Graphen wieder kompakt als eine Gleichung ausgedrückt werden kann.

2.2.1.3 Unifikation von Merkmalsstrukturen

Der Zentrale Begriff für die Unifikation von Merkmalsstrukturen ist die Subsumption.

Definition 2.12 *Subsumption zweier Merkmalsstrukturen:*

Eine Merkmalsstruktur F_1 subsummiert eine Merkmalsstruktur F_2 gdw. wenn für jedes Merkmal f_1 in F_1 ein Merkmal f_2 in F_2 existiert, sodaß der Wert von f_1 den Wert von f_2 subsummiert.

Uninstanziierte Variablen subsummieren alles.

Atome subsummieren nur sich selbst.

Die Subsumption etabliert also eine partielle Ordnung auf Merkmalsstrukturen. Eine allgemeinere MS subsummiert immer eine speziellere MS.

Das Resultat der Unifikation zweier Merkmalsstrukturen F_1 und F_2 läßt sich dann einfach definieren als die kleinste Merkmalsstruktur F_3, die sowohl von F_1 als auch von F_2 subsummiert wird.

2.2.1.4 Disjunktion

Disjunktion innerhalb von Merkmalsstrukturen ist das wichtigste Ausdrucksmittel bei der sprachlichen Beschreibung. Einfache Unifikationsformalismen können nur Atome disjunktiv beschreiben. Inzwischen hat sich jedoch die Verwendung der vollen Disjunktion durchgesetzt. Ihre Behandlung ist ausführlich in Eisele & Dörre 1990 [25] beschrieben.

Bei der vollen Disjunktion kann an jeder Stelle einer Merkmalsstruktur ein "oder" eingefügt werden. Für verschiedene Varianten von Unifikationsformalismen, die Beschreibungen unter Verwendung aussagenlogischer Konnektive erlauben, wurde gezeigt, daß die Subsumption von Merkmalsstrukturen NP-vollständig ist (u.a. Johnson 1987 [47] und Smolka 1988 [88]).

Eine Variante der Disjunktion ist die verteilte Disjunktion (Dörre & Eisele 1989 [21] u.a.), welche es erlaubt, innerhalb einer Merkmalsstuktur Abhängigkeiten zwischen Mitgliedern verschiedener Disjunktionen festzulegen. Bezüglich der prinzipiellen Verarbeitungskomplexität führt dies nicht zu Unterschieden.

2.2.1.5 Funktionen und Relationen

In jüngerer Vergangenheit sind im Kontext der Verwendung von HPSG ([72]) in der Sprachverarbeitung vermehrt Formalismen entwickelt worden, die Relationen (und seltener auch Funktionen) in Merkmalsstrukturen einbetten. Während einer Unifikation werden die enthaltenen Relationen ausgewertet und beeinflussen so das Ergebnis der Unifikation.

Unifikationsformalismen, die eine freie Verwendung von Relationen über Mermalsstrukturen erlauben, sind turingmächtig.

Während Unifikationsformalismen ohne Relationen üblicherweise als ADT von einer Programmiersprache aus verfügbar sind, konstituieren solche mit Relationen, wie z.B. STUF3, meist selbst bereits eine volle Logikprogrammiersprache in der Art von PROLOG.

In der Implementierung solcher Systeme kommt zumeist keine statische Repräsentation von Disjunktionen vor. Hingegen werden Relationen von einer Inferenzmaschine bearbeitet, die durch Backtracking Disjunktionen intern als Nichtdeterminismen behandelt.

Arbeiten in diesem Bereich sind u.a. Dörre & Seiffert 1991 [22] und Backofen 1989 [7].

2.2.1.6 Typen

Zusätzlich zur "normalen" Unifikation von Merkmalsstrukturen gibt es die Typenunifikation. In neueren Unifikationsformalismen gewinnt die Typenkomponente zunehmend an Gewicht. Grundsätzlich hat bei einem typisierten Formalismus jede Merkmalsstruktur einen Typ. Die Typen bilden selbst wieder einen Verband, der vom Benutzer in Form von Subsumptionsrelationen angegeben werden kann. Verwendete Typsysteme lassen sich vor allem darin unterscheiden, wie eng die Typen an die eigentlichen Merkmalsstrukturen geknüpft sind. Obwohl es auch Zwischenstufen gibt, lassen sich grob drei Arten von Typsystemen unterscheiden, nämlich schwache Typsysteme, Typsysteme mit *Angemessenheitstest* und *rekursive Typinferenz*.

Bei schwachen Typsystemen ist mit einem bestimmten Typ keine Forderung verbunden, daß eine ihm zugeordnete Merkmalsstruktur Bedingungen erfüllen muß. Indem die Subsumptionsbeziehungen vor der Ausführung von Unifikation für Typen bereits feststehen, kann in einem Vorverarbeitungsschritt ein effizient kodierter Typverband hergestellt werden. Die Typunifikation ist dann wesentlich effizienter als die Unifikation der Merkmalsstrukturen. Bei einer Unifikationsoperation werden dann zunächst die Typen unifiziert. Wenn diese Operation gelingt, wird die Merkmalsstruktur unifiziert. Die sich ergebende Merkmalsstruktur erhält den Ergebnistyp der Typenunifikation. Indem möglichst viel Information in Typen kodiert ist, wird so eine effiziente Verarbeitung erreicht. Der in dieser Arbeit verwendete Formalismus ([29], [30]) bietet die Möglichkeit, auf diese Weise zu verarbeiten, obwohl er auch mit Angemessenheitsbedingung verwendet werden kann.

Durch HPSG [72] hat der Ansatz Verbreitung gefunden, Typen mit bestimmten Merkmalen und Werten der Terme zu verknüpfen, denen sie zugeordnet sind. Formal ist ein solches Typsystem in Carpenter 1991 [14] beschrieben. Eine *Appropriatness-Funktion* liefert für einen Typ einer MS und ein Merkmal den größten Typ des Wertes dieses Merkmals zurück. Eine Unifikation zerfällt schließlich in die Typunifikation, die Unifikation der MS und die Überprüfung der Typbedingungen. In einer Typhierarchie wird ein Merkmal nur noch an einer Stelle eingeführt und wird von dieser Stelle aus an die Untertypen weitervererbt.

Bei rekursiver Typinferenz wird eine ganze Merkmalsstruktur fest mit einem Typnamen verbunden. Eine solche MS kann alles enthalten, was der Formalismus erlaubt, also auch Disjunktionen, Koreferenzen usw. Der volle Rückschluß von einer MS auf einen Typ ist möglich. Eine operationale Semantik für einen solchen Formalismus wird in Emele & Zajak 1990 [26] beschrieben. Implementiert ist ein solcher Formalismus unter dem Namen *Typed Feature System* im Projekt POLYGLOSS.

In einem Kompilationsschritt wird für jeden Typ wirklich eine Merkmalsstuktur gebildet und abgelegt. Eine Kopie dieser MS wird dann bei jeder Unifikation eines Terms dieses Typs mit unifiziert. Im Unterschied zur Vorgehensweise von Carpenter 1991 [14] werden durch Typdefinitionen hier Merkmale erzeugt. Durch die Angemessenheitsbedingungen wird nur ein Scheitern der Unifikation bei Verletzung der Angemessenheit herbeigeführt.

2.3 Standard Parsingverfahren für natürliche Sprache

In den letzten 25 Jahren wurden zahlreiche Verfahren für das Parsing von kontextfreien Sprachen entwickelt. Die klassischen Vertreter sind dabei der *Earley-Algorithmus* (EA) [24], der *Cocke-Younger-Kasami-Algorithmus* (CYK) und der *Shift-Reduce-Parser* für LR(k)-Grammatiken (LRP) [1]. Zwei Parsingverfahren haben insbesondere in der Verarbeitung natürlicher Sprache Verwendung gefunden: Der *Tomita-Parser* oder auch *Generalisierter LR-Parser* (GLRP), eine Erweiterung des LRP auf mehrdeutige CFG, und der *Aktive Chartparser* (ACP), eine Generalisierung des EA bezüglich der Reihenfolge, in welcher der Suchraum beschritten wird.

Neben diesen Verfahren sind in der sprachorientierten KI zahlreiche Verfahren entwickelt worden für spezielle Typen von Grammatiken, die nicht auf CFG beruhen. Letztere sind hier jedoch von geringerem Interesse.

Sowohl für GLRP als auch für ACP sind Erweiterungen für Lattice-Parsing ausgearbeitet worden, daher sind die beiden Verfahren im Rahmen dieser Arbeit vor allem von Interesse. GLRP für Lattices findet sich z.B. bei Tomita 1986 [92] oder Wright 1990 [98]. ACP als Latticeparser (mit Unifikationsgrammatiken) wurde in Andry & Thornton 1991 [5], Görz 1988 [37] und Weber 1992 [95] vorgestellt.

2.3.1 GLRP

Obwohl die GLRP in dieser Arbeit nicht zentral sind, werden sie hier kurz eingeführt. In Abschnit 5 wird in Breite auf statistische GLRP Bezug genommen, da aus diesem Bereich wichtige Arbeiten im Bereich der Kopplung von Statistik und Unifikationsgrammatik kommen.

Generalisierte LR-Parser sind von verschiedenen Autoren vorgeschlagen worden, u.a. Lang 1974 [56], Tomita 1985 [91] und Schabes 1991 [78]. Grundsätzlich basieren diese Parser auf aus Grammatiken vorkompilierten nichtdeterministischen Push-Down-Automaten, für die effiziente Verarbeitungstechniken eingeführt werden. Der klassische GLRP aus [91] ist wie folgt aufgebaut:

1. Erstelle eine LR(1)-Tabelle, bei der jedes Feld eine Menge von Schritten anstatt einen Schritt enthält,. z.B. mithilfe des klassischen Verfahrens aus Aho & Ullmann 1972 [1]

2. Verwende die Tabelle als NPDA mit einem Graph als Stack. Für jeden Shift-Reduce Konflikt verzweige den Stack. Wenn zwei Verzweigungen des Graph-Stacks den gleichen Zustand in der LR-Tabelle haben, verschmelze sie zu einem Zweig.

3. Baue während des Terminal-Einlesens, falls ein abgearbeitetes Symbol vom Stack gepoppt wird, einen geteilten Wald von Syntaxbäumen auf. Hierbei werden identische Teilbäume nur einmal dargestellt (*Subtree Sharing*) und verschiedene Teilbäume mit gleichem Label zusammengefaßt (*Local Ambiguity Packing*).[14]

Tomita 1987 [93] präsentiert eine Erweiterung auf PATR-artige[15] Unifikationsgrammatiken.

Die Erweiterung des Verfahrens auf Unifikationsgrammatiken setzt voraus, daß eine Abbildung von der Unifikationsgrammatik auf eine kontextfreie Grammatik vorgenommen wird. Dies ist nötig, um die Shift-Reduce-Tabelle, die den Automaten treibt, zu erstellen. Dies kann im einfachsten Fall für Formalismen ohne Typisierung wie folgt geschehen.

Die Merkmale, die in einer Unifikationsgrammatik vorkommen, zerfallen in zwei Teilklassen: *infinite Merkmale*, die unendlich viele Ausprägungen von Werten haben können[16], und *finite Merkmale*, die nur endlich viele mögliche Werte haben können.

Für eine gegebene Unifikationsgrammatik UG, erstelle man nun eine Grammatik UG', die wie folgt aus UG entsteht:

1. Für jeden Lexikoneintrag a aus UG erhält UG' einen Lexikoneintrag a', wobei die Merkmalsstruktur von a' aus der Merkmalsstruktur von a durch Entfernen aller Infiniten Merkmale entsteht.

2. Für jede Regel b aus UG erhält UG' eine Regel b', wobei b' aus b entsteht, indem aus allen Merkmalsstrukturen von b die infiniten Merkmale entfernt werden.

Jedes Merkmal einer Grammatik entspricht einer zusätzlichen Beschränkung über den möglichen Ableitungen der Grammatik. Daher vergrößert sich die Sprache einer Grammatik durch Entfernen von Merkmalen monoton. UG' erzeugt also eine Obermenge der Sprache von UG.

Aus UG' kann eine kontextfreie Grammatik UG" erzeugt werden durch Bildung aller Ausprägungen von Regelinstanzen. Aus UG" wird schließlich die LR-Tabelle erzeugt (w.o.).

Der Tomita-Parser für UG kann nun auf zwei verschiedene Weisen betrieben werden. Entweder werden die Regeln aus UG jedesmal bei einem Schritt des Parsers mitausgeführt. Hierbei werden durch fehlschlagende Unifikationen eventuell Pfade abgebrochen. Oder aber der Parser findet alle Analysen mit UG" und bildet dann

---

[14] Wenn zwei in einer Tabelle dargestellte Bäume einen gemeinsamen Teilbaum besitzen, so spricht man von *Subtree Sharing*, wenn der gemeinsame Teilbaum nur einmal in der Tabelle dargestellt ist. Wenn ein Parsingverfahren einen Teilparsbaum findet mit einem Grammatiksymbol X als Wurzel und in der Analysetabelle existiert schon ein anderer Teilbaum mit gleicher Überdeckung und gleicher Wurzel, so spricht man von *Local Ambiguity Packing* wenn dann kein neuer Eintrag in die Tabelle erfolgt, sondern die zweite durch die erste Analyse mit dargestellt wird. Beide Techniken sind gebräuchlich und verantwortlich für die günstige kubische Schranke beim Parsing von CFG.

[15] PATR ist eine frühre Unifikationsgrammatik. In PATR wurden zu einer kontextfreien Grammatik einfach an jedes nichtterminale Symbol der Grammatik Merkmale und Werte angehängt.

[16] Iterierbare Merkmale haben Merkmals-Wert Strukturen als mögliche Werte, bei welchen sie gegebenenfalls eingebettet wieder selbst in einem Pfad erscheinen können. infinite Merkmale können iterierbare Merkmale einbetten, finite Merkmale können dies nicht.

die Analysen mit UG nach, wobei nicht alle Analysen überleben. Die nachgezogenen Unifikationen wirken als nachträglicher Filter.

2.3.2 ACP

Aktives Chartparsing ist eine Generalisierung des Earley-Verfahrens[17] bezüglich der Reihenfolge, in der Items erzeugt werden, ergo der Richtung, in welcher der Suchraum durchschritten wird. Eine Menge von Knoten und Kanten heißt *Chart*. Die Grundelemente hierbei sind:

Chartknoten: Abstrakte Zeitpunkte

Chartkanten: Paare von Knoten (von,bis), und eine *dotted Rule*, entsprechend den Items des Earley-Parsers.

Aktive Kante: Kante, bei welcher der Punkt der *dotted Rule* nicht am rechten Rand der Regel liegt. Die Regel ist also noch nicht abgearbeitet.

Passive Kante: Kante, bei welcher der Punkt der *dotted Rule* am rechten Rand der Regel liegt. Die Regel ist also abgearbeitet.

Die Grundoperation der Chartanalyse ist die *Fundamentale Regel*. Sie entspricht in ihrer Funktion einem Schritt des Completers des Earley-Parsers. Hier für den kontextfreien Fall:

Fundamentale Regel :
Falls A eine aktive Kante ist, B eine passive Kante ist und A.bis = B.von, und A.next-daughter[18] = B.mother[19], dann füge C in die Chart ein mit C.von := A.von, C.bis := B.bis und C.rule := A.rule[20], mit dem Punkt eine Position nach rechts geschoben.

Der ACP kann im Bottom-Up bzw. im Top-Down-Modus arbeiten. Dabei werden jeweils aufgrund von Erwartungen aktiver Kanten oder aufgrund von gefundenen passiven Kanten neue leere aktive Kanten eingefügt.

Seek-Up :
Falls eine passive Kante A in die Chart eingefügt wird, dann füge für jede Grammatikregel r, sodaß A.mother = linkeste Tochter von r, eine neue aktive Kante B in die Chart ein mit: B.von := A.von, B.bis := A.von und B.rule := r, mit Punkt ganz links, falls eine solche Kante nicht schon existiert.

Seek-Down :
Falls eine aktive Kante A in die Chart eingefügt wird, dann füge für jede Grammatikregel r, sodaß A.next-daughter = Mutter von r, eine neue aktive Kante B in die Chart ein mit: B.von := A.bis, B.bis := A.bis und B.rule := r, mit Punkt ganz links, falls eine solche Kante nicht schon existiert.

[17] Der Earley Parser ist sozusagen der direkte Vorläufer der Aktiven Chartparser. Das klassische Verfahren wird hier aus Platzgründen nicht eingeführt. Der Leser, der nicht mit dem EA vertraut ist, findet eine gute Darstellung in Aho & Ullmann 72 [1].

[18] X.next-daughter gibt für die Kante X die Kategorie zurück, die auf der rechten Seite der zur Kante gehörenden Regel direkt rechts von Punkt steht.

[19] X.mother gibt für die Kante X die Kategorie zurück, die auf der linken Seite der zur Kante gehörenden Regel steht.

[20] X.rule gibt für die Kante X die dotted Rule zurück.

Seek-Down entspricht dem Prediktor-Schritt des Earley-Parsers. Seek-Up kann als "Bottom-Up"-Version des Prediktors bezeichnet werden. Die Suchstrategie des ACP wird über eine Agenda verwaltet, die Paare von aktiven und passiven Kanten enthält, welche wie folgt auf die Agenda gelegt werden.

Agenda-Push :

> Falls eine aktive Kante A eingefügt wird, dann lege alle Paare (A,B) auf die Agenda, sodaß B passiv ist und A.bis = B.von.
>
> Falls eine passive Kante B eingefügt wird, dann lege alle Paare (A,B) auf die Agenda, sodaß A aktiv ist und A.bis = B.von.

Falls die Agenda als Stack implementiert ist, resultiert eine Tiefensuche, während eine Queue der Breitensuche des Earley-Verfahrens entspricht. Eine sortierte Agenda implementiert eine Bestensuche, wie sie z.B. für Dynamische Programmierung im Chart-Parser[21] benötigt wird. Der ACP startet, indem eine aktive Startkante in den ersten Knoten der Chart eingefügt wird. Danach wird die Fundamentale Regel nacheinander auf die Paare der Agenda angewendet.

Im Bottom-Up-Modus ist keine Veränderung des Parsingschemas nötig um Unifikationsgrammatiken mit dem ACP zu verarbeiten. Sämtliche Tests über Kategoriensymbole werden selbstverständlich durch Unifikationen ersetzt. Falls der Parser im Top-Down-Modus arbeitet, wird beim Einfügen neuer aktiver Kanten ein spezieller Restriktionsmechanismus[22] angewendet und der Redundanztest durch Kategorienidentität wird durch einen Subsumptionstest ersetzt. Auf den Restriktionsmechanismus wird in Kapitel 3 noch ausführlicher eingegangen. Dort wird auch eine Methode zum Vorherberechnen dieses Mechanismus vorgestellt.

Subtree Sharing, wie bei der Darstellung eines verteilten Waldes beim Tomita-Parser, ist dem ACP inhärent. Die Chart funktioniert als Verwaltung von Parses für Teilketten der Eingabe. Weitere verschiedene Parses für Teilketten, die die erste Teilkette enthalten, verweisen schließlich in der Chart auf die Kante für den gemeinsamen Anteil, teilen also diesen Teilbaum.

Durch einen zusätzlichen Redundanztest beim Einführen von passiven Kanten wird eine Entsprechung zum *Local Ambiguity Packing* erreicht. Im kontextfreien Fall wird eine Analysekante nicht eingefügt wenn eine Kante der gleichen Kategorie über der gleichen Teilkette bereits existiert. Letztere steht danach für beide Ableitung. So werden im Chartparser mehrere Ableitungen für ein Teilergebnis gepackt dargestellt.

Sowohl beim GLRP als auch beim ACP ist *Subtree Sharing* bei Unifikationsgrammatiken entweder mit Kopieren oder aber mit Verwalten von Kontexten von Merkmalsstrukturen verbunden.

Bei beiden Parsern wird für Unifikation bei *Local Ambiguity Packing* ein Subsumptionstest durchgeführt und die generellste MS weiterverfolgt.

Eine Merkmalsstruktur steht dann für beide Ableitungen. Indem nur die generellste MS weiterverfolgt wird, ist garantiert, daß im Falle eines Fehlschlagens einer Unifikation, die entsprechende Operation auch mit einer spezielleren MS fehlgeschlagen wäre.

[21] Siehe z.B. Ney [65] oder Päseler [67] für DP-Parsing.
[22] Vergleiche Shieber 1985 [86].

2.3.3 Schnittstellen zwischen Parser und Decoder

Wie oben bereits eingeführt kommen verschiedene Möglichkeiten der Kopplung von Worterkennung, Wortkettenerkennung und Wortkettenparsing in Frage. In der Standardarchitektur[23] sind Wort- und Wortkettenerkennung ineinander verschränkt, während das Parsing abgekoppelt durchgeführt wird. Hierbei gibt es zwischen Erkennung und Parsing drei Schnittstellentypen, die Wortlattice, der verbundene Wortgraph und N-beste-Ketten.

Zunächst bieten wir jedoch der Vollständigkeit halber eine Definition für Worthypothesen an, die durch die gesamte Arbeit verwendet werden kann.

Definition 2.13 *Eine Worthypothese ist ein 6-Tupel, (from, to, key, score, info, predecessor), wobei:*

from *ist der Anfangszeitpunkt einer Worthypothese. In dieser Arbeit ist dies die Nummer des Frames, in welcher das korrespondierende Wortmodell vom Decoder gestartet wurde.*

to *ist der Endezeitpunkt einer Worthypothese. In dieser Arbeit ist dies die Nummer des Frames, in welchem der Decoder für das korrespondierende Wortmodell einen Endezustand oberhalb des Beams findet.*

key *ist die Wortform der Worthypothese und der Zugriffsschlüssel zum Lexikon der vom Parser verwendeten Grammatik.*

score *ist eine Bewertung der Worthypothese. Bei bottom up gesendeten Hypothesen ist dies die akustische Bewertung, also die Modellwahrscheinlichkeit der Framefolge, die durch einen HMM-Pfad überspannt wird.*[24]

info *ist ein Feld daß ein gesetztes Flag enthalten kann. Bei bottom up Kopplungen wird dieses Feld nicht beachtet. Bei top down Kopplungen von Decoder und Parser findet es Verwendung um diverse Protokolle zu implementieren*[25]*.*

predecessor *ist der beste Vorgänger, den eine Worthypothese hat. Bei Hypothesen, die in Frame 0 beginnen, ist dies ein BEGIN-Marker.*

2.3.3.1 N-Beste Wortketten

Bei der N-Beste-Schnittstelle zwischen Erkennung und Parsing werden dem Parser die N bestbewerteten Hypothesen von Wortketten übergeben, welche den gesamten Äußerungszeitraum überspannen.

Die Vorteile dieses Vorgehens sind:

- Jede Art von Parser kann verwendet werden. Eine spezielle Entwicklung für die Integration mit der Speech-Komponente ist nicht nötig.

[23] Wir meinen hier nicht inkrementelle Architekturen, mit einem bottom up Datenfluß. Z.B. HARC (BBN), ATIS (MIT) oder die Architektur von ASL-Süd.

[24] Die Details der Bewertungen werden in den folgenden Kapiteln ausführlich diskutiert.

[25] Siehe Abschnitt 4.

- Falls die Anwendung eine gewisse Größe nicht übersteigt, bezüglich Anwendungsdomäne, Sprecherabhängigkeit, Vokabular und akustischer Umgebung, ist mit einer guten Worterkennungsrate zu rechnen, also mit einem hohen Rang der korrekten Wortkette, bzw. mit einem hohen Anteil korrekter bester Ketten. In diesem Fall ist die Schnittstelle sehr effizient.

Die Nachteile sind:

- Falls, z.B. bei Lockerung von Systemrestriktionen, die Worterkennungsrate sinkt, kann es vorkommen, daß die korrekte Wortkette einen niedrigen Rang hat. In diesem Fall müssen viele falsche Ketten teilanalysiert und schließlich doch verworfen werden.
- Viele Hypothesen von Wortketten unterscheiden sich nur in einem Wort. Das bedeutet, daß der identische Rest zweier Hypothesen zweimal vom Parser analysiert werden muß. Eine Strukturteilung identischer Teilfolgen verschiedener Hypothesen ist nicht möglich.

Es gibt inzwischen eine stattliche Anzahl von Algorithmen, um N-Beste-Ketten durch HMMs und N-Gramme effizient zu berechnen. Unterschiede liegen hier vor allem in der Art und Weise, in welcher der Suchraum beschnitten wird.

2.3.3.2 Verbundene Wortgraphen

Definition 2.14 *Ein verbundener Wortgraph W ist eine Menge von Worthypothesen w_1 bis w_n für die eine Verbundenheitsbedingung gilt.*

Definition 2.15 Verbundenheitsbedingung *:*
Für jede Worthypothese w_j , $1 \leq j \leq n$, eines verbundenen Wortgraphen W gibt es mindestens ein w_i und ein w_k, sodaß $from(w_i)=to(w_j)$ und $from(w_j)=to(w_k)$. Am Äußerungsbeginn endet per Definition w_0, am Äußerungsende beginnt analog per Definition w_{n+1}.

Wortgraphen lassen sich direkt als Chart darstellen. In der Chart werden konkurrierende Analysehypothesen über dem selben Zeitraum als zwei verschiedene passive Kanten dargestellt, welche die gleichen Anfangs- und Endknoten besitzen. Diese Vorgehensweise wird im Falle von verbundenen Wortgraphen als Eingabe des Parsers auf die terminalen Kanten[26] ausgedehnt. Die Chartknoten korrespondieren nun nicht mehr mit Wortgrenzen, sondern als abstrakte Zeitpunkte[27] mit Hypothesen über Wortgrenzen. Der Analysealgorithmus der Chartanalyse arbeitet unverändert.

Auch der GLRP für Lattices, der von Tomita 1986 [92] vorgeschlagen wurde, ist eigentlich ein Parser für verbundene Wortgraphen. Das Problem, ob zwei Worthypothesen wirklich adjazent sind, wird hier ausgeklammert durch die Annahme von Prädikaten, die diese Information zur Laufzeit liefern und deren Funktion nicht weiter geklärt wird. Während des Einlesens ruft der Tomita-Parser diese Prädikate

[26] Mit *terminaler Kante* bezeichnet man eine passive Kante, die direkt aus dem Lexikonzugriff resultiert, also nicht durch eine Regelanwendung zustandekommt.
[27] .., die durchaus mit konkreten Zeitpunkten identifiziert werden können, wie z.B. in der Implementation des Chartparsers im BMFT-Projekt ASL-Nord. Vergl. Weber 1992 [95].

auf und Worthypothesen, die von links nach rechts nicht erreichbar sind, werden dabei ignoriert. Somit wird also nur der "verbundene Teil" einer Lattice überhaupt eingelesen.

N-Beste-Ketten sind ein Spezialfall von Wortgraphen. Man erhält aus der ersten Darstellung die zweite, indem ein gemeinsamer Anfangs- und Endknoten hinzugefügt wird. Der resultierende Wortgraph kann dann jedoch doppelte Worthypothesen enthalten, während die Grundidee der Wortgraphendarstellung ja gerade die effiziente Strukturteilung auf lexikalischer Ebene ist.

2.3.3.3 Lattices

Definition 2.16 *Eine Lattice ist eine Menge von Worthypothesen über einem Abtastzeitraum.*

Die Worthypothesen können hier zeitlich überlappen oder Lücken aufweisen. *Lattice* ist also die allgemeinste Definition einer Schnittstelle. Eine einzige Worthypothese kann ebenso eine Lattice sein wie das gesamte Lexikon, hypothetisiert über jeder möglichen Segmentfolge. Wortgraphen sind Spezialfälle von Lattices, die der *Verbundenheitsbedingung* genügen.

Parsing setzt einen abstrakten Begriff der Adjazenz voraus. Daher ist vor oder während des Parsing über Lattices eine jeweilige Entscheidung nötig, ob Überlappungen oder Lücken beachtet werden oder nicht. Es gibt diesbezüglich zwei Vorgehensweisen, die beide innerhalb der Chart realisiert werden können. Chien et al. 1990 [16] schlagen einen Vorverarbeitungsschritt vor, der zusätzliche *Jump-Edges* einführt, mit denen der Parser kurze Zeitintervalle von Overlaps oder Gaps überspringen kann. Dieses Verfahren hat einen Vor- und einen Nachteil:

Vorteil: Das Verfahren kann mit variablen Zeitintervallen umgehen. Der Zeitraum einer Überlappung oder einer Lücke, der in einem bestimmten Fall zweier Worthypothesen noch toleriert wird, kann von Fall zu Fall verschieden sein, z.B. abhängig von Phonemeigenschaften[28]

Nachteil: Die ganze Lattice muß voll verfügbar sein. D.h., das Verfahren ist nicht geeignet, in Verbindung mit zeitsynchronen, parallelen Architekturen eingesetzt zu werden.

Ein anderes, starreres Verfahren, welches jedoch für interaktives Parsing geeignet ist, wurde im ASL-Demonstrator 92 implementiert und ist bei Weber 92 [95] beschrieben. Hierbei werden von vornherein die Knoten der Chart konkreten Zeitpunkten mit festen Intervallen zugeordnet. Jede Hypothese einer Lattices wird beim Einfügen in das vorgegebene Raster eingepaßt. Beim Parsing werden dann in der Fundamentalen Regel auch solche Hypothesenpaare betrachtet, die um k Knoten[29] getrennt sind oder überlappen. Dieses Verfahren ist jedoch "teurer" als das Verfahren von Chien et al. 1990. Doppelarbeit kann bei einem starren Verfahren nicht vermieden werden. In Abschnitt 4.6.4 dieser Arbeit wird ein inkrementelles Verfahren vorgestellt, das sich optimal verhält.

[28] Falls von zwei Wortkandidaten der erste auf den gleichen Nasal endet, mit dem der zweite beginnt, ist der Zeitraum, in welchem das erste Wort enden kann und das zweite starten kann sehr viel größer als z.B. bei der Kombination eines Verschlusslautes und eines Vokals.

[29] Der Wert für k muss in Abhängigkeit von der die Knoten betreffenden Granularität der Zeitintervalle vorher bestimmt werden.

Kapitel 3

Ein zeitsynchroner interaktiver ACP für Worthypothesen

Zeitsynchrones und interaktives Parsing ist erst in wenigen Arbeiten zur Speech- und Language-Schnittstelle untersucht worden. Die bisherigen Arbeiten, die eine enge Verzahnung vorschlagen, nutzen jedoch vergleichsweise einfache Formalismen für die symbolische Sprachbeschreibung, die für eine Anwendung mit größerer Abdeckung nicht geeignet sind. Murveit et. al. 90 [62] und Dupont 93 [23] stellen z.B. Arbeiten vor, in denen enge Kopplungen zwischen Decoding und Parsing mit endlichen Automaten und kontextfreien Grammatiken durchgeführt werden. Die Ergebnisse dieser Arbeiten sind vielversprechend. Dennoch bleiben sie im Bereich der Spielanwendungen, denn der Aufbau einer semantischen Struktur ist mit solchen Formalismen nicht möglich. Dem Stand der Kunst im Bereich Sprachverstehen entspricht die Kopplung mit einer unifikationsbasierten Grammatik. Hier erst bleibt die Möglichkeit offen, in Zukunft eine ebenfalls inkrementelle Kopplung zwischen Syntax und Semantik, bzw. Transfer zu realisieren.

Im Vergleich zu grammatischen Operationen beim Parsing von kontextfreien Grammatiken[1] ist die Unifikationsoperation teuer bezüglich Rechenzeit und Speicherbedarf. Neben einer effizienten Implementation wird die Einbeziehung von Bewertungen und Pruning-Mechanismen also entscheidend.

In diesem Kapitel wird ein ACP beschrieben, der zeitsynchron mit einem Beam Decoder arbeiten kann und mit dem es möglich ist, Unifikationsgrammatiken so effizient zu verarbeiten, daß eine enge Kopplung möglich wird. Der Schlüssel hierfür ist eine enge Verzahnung von Suche und Analyse innerhalb des Parsers.

Die Besonderheiten des *LR-ACP* (Links-Rechts Inkrementeller Activer Chartparser) sind im folgenden kurz zusammengestellt.

- Alle Hypothesen eines Wortgraphen werden von links nach rechts in eine Chart eingelesen und gleichzeitig verarbeitet.
- Die SEEK-DOWN-Operation ist vollständig vorverarbeitet.

[1] Der billige Vergleich zweier Namen.

- Der ACP nutzt ein statistisches Modell und eine Unifikationsgrammatik simultan.
- Der Parser bewertet jede Aktion vorher durch das statistische Modell und verwirft schlecht bewertete Aktionen.
- Die Vorwärtssuche des Parsers ist eine Strahlensuche (Beamsearch)
- Die Rückwärtssuche des Parsers (Completerschritt) ist ebenfalls eine Strahlensuche.
- Der Parser ist erweiterbar hinsichtlich diverser Interaktionsstrategien mit anderen Modulen.[2]
- Die Metrik ist erweiterbar zur Miteinbeziehung weiterer Quellen statistischer Modellwahrscheinlichkeiten.[3]

Die Grundingredenzien für diesen speziellen Parser für Worthypothesen kommen von Arbeiten über Chartparsing, wie Görz 1988 [37], Shieber 1985 [86] und Thompson 1990 [90]. Starke Einflüsse kommen von Ney 1991 [65] zur dynamischen Programmierung im Chartparser und Arbeiten zur Kopplung wie Päseler 1988 [67]. Aber auch die Sichtweisen von Briscoe 1987 [12] oder Altmann 1990 [3] zur Architektur von integrierten Speech–Language–Systemen sind deutlich sichtbar. Auf Arbeiten zur Verwendung von grammatischen Ableitungswahrscheinlichkeiten wie Baker 1979 [8], Fujisaki 1984 [32] oder Jelinek 1992 [44] ist aufgebaut worden.

3.1 Worthypothesen in der Chart

Eine Chart läßt sich durch geringe Änderungen in der Verwaltung auf die Darstellung von Worthypothesen-Lattices erweitern. Für nicht–inkrementelle Verfahren wurde dies bereits bei verschiedenen Autoren vorgestellt (u.a. Görz 1988 [37], Chien et. al. 1990 [16], Thompson 1990 [90], Ney 1991 [65]) oder Brietzmann 1992 [11].

3.1.1 Darstellung von Konkurrenz

Eine aktive Chart stellt konkurrierende Analyseergebnisse als zwei Kanten mit den gleichen Anfangs- und Endknoten dar. Bei der Verarbeitung von einfachen Wortketten repräsentieren Knoten abstrakte Zeitpunkte zwischen je zwei Wörtern.

Um Lattices darzustellen, verankern wir die Knoten der Chart an konkreten Zeitpunkten. Konkurrierende Worthypothesen werden als zwei Kanten mit identischen Anfangs- und Endknoten dargestellt.

In Abbildung 3.1 sind die w_i Worthypothesen und die p_j Phrasenhypothesen. Bezüglich ihrer Repräsentation sind die beiden (fast) nicht unterschieden.

Bei Chien et al. 1990 [16] wird ein Verfahren vorgestellt, um konkrete Zeitpunkte einer Wortlattice auf abstrakte Chartknoten abzubilden – ein Problem, daß in dieser Arbeit als *Time Map* bezeichnet wird. Leider ist dieses Verfahren nur anwendbar, wenn die Lattice als ganzes vorliegt, was bedeutet, daß es für ein inkrementelles, zeitsynchrones Parsing nicht einsetzbar ist. In Abschnitt 4.6.4 wird ein zeitsynchrones Verfahren für einen Time Map entwickelt.

[2] Siehe Abschnitt 4.

[3] Siehe Abschnitt 5.

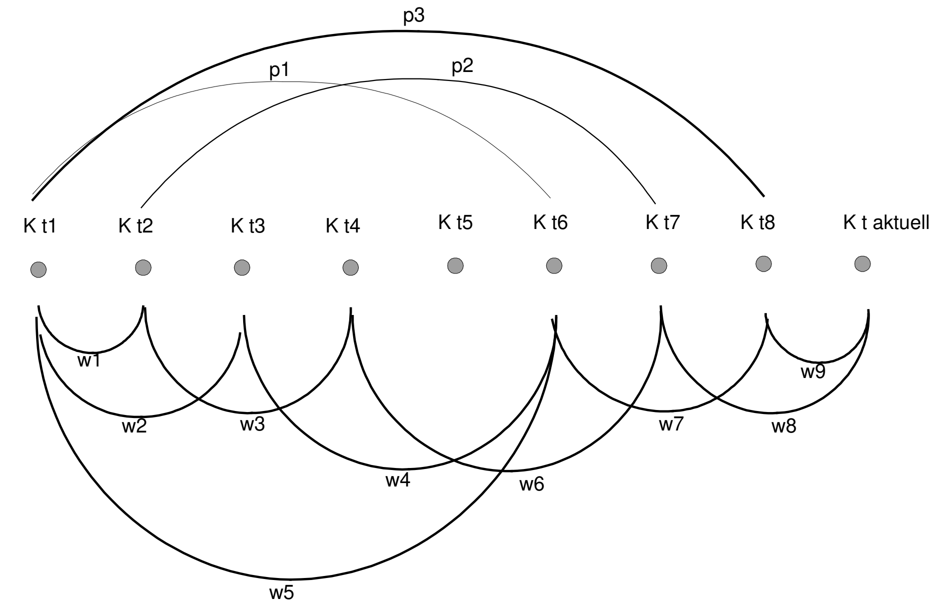

Abbildung 3.1: Die Chart mit Wort- und Phrasenhypothesen

3.1.2 Komplexität von Wortgraphenparsing in der Chart

Von Sheil 1976 [85] ist bekannt, daß Chartparsingverfahren, sofern *Structure Sharing* und *Local Ambiguity Packing* verwendet werden, eine Platz- und Zeitkomplexität für CFGs von $O(n^3)$ besitzen.[4] n ist dabei die Anzahl der Wörter in der Eingabeliste.
Wir betrachten nun das Verhalten eines ACP bezüglich der Verarbeitung von Mengen von Worthypothesen – also Lattices, wie im Kapitel 2.3.3.3 beschrieben.

3.1.2.1 Obere Schranke abhängig von Frames

Wir betrachten einen Beam-Decoder wie in Kap. 2.1.4 beschrieben. Ein solcher Decoder wurde für die Experimente in dieser Arbeit verwendet[5].

Um einen Eindruck zu erhalten, wie komplex das Parsing von Lattices bzw. Wortgraphen ist, vernachlässigen wir zunächst einmal die statistische Komponente und konzentrieren uns auf die generativen Anteile der Worterkennung und des Parsing.

Die Markovmodelle, die im Decoder als Wortmodelle verwendet werden, sind zeitsynchron, d.h. es gibt nur Zustandsübergänge von i nach j, sodaß entweder gilt:

zyklisch: $i = j$, oder

azyklisch: $i \prec j$

Die Folge von Zuständen eines HMMs kann also ebenso durch eine Menge von CF-Regeln erzeugt werden, wobei für ein Wortmodell eine Regel genau die Zustände des Modells erzeugt (der azyklische Fall). Für jeden Zustand gibt es zusätzlich eine Regel, der Form $i \rightarrow ii$, für den zyklischen Fall. [6] Grundsätzlich ist diese Vorgehensweise immer möglich, da endliche Automaten durch das stärkere formale Modell der CFGs dargestellt werden können.

Die Emissionswahrscheinlichkeiten werden ebenfalls durch unäre Grammatikregeln dargestellt, solcherart, daß für jede Beobachtung jeder Zustand eines Wortmodells erzeugt werden kann.

Eine solche Grammatik kann jetzt über jeder Kette von Frames jedes Wort des Lexikons ableiten. Ein Decoder erzeugt genau dann die gleichen Worthypothesen, wenn er ohne Beam arbeitet. In diesem Fall erzeugt er alle Wortformen des Lexikons über jedem Abschnitt des Signals. Wir nennen diese Lattice L_{max}. L_{max} ist im übrigen ein verbundener Wortgraph. Durch die Beschränkung des Chartparsing auf $O(n^3)$ ist sichergestellt, daß L_{max} innerhalb dieser Zeit hergestellt werden kann. n ist hier jedoch nicht mehr die Länge der Wortkette, sondern die Anzahl der Frames, die ein Decoder als Eingabe verwendet.

[4] Dies gilt im Übrigen nur für das reine Parsing. Falls zum Zwecke des Entpackens einer Chart Zeiger auf Tochterkanten verwendet werden, können die Listen solcher Zeiger schon wieder exponentiell groß werden. Dies ist insbesondere bei statistischen Parsing mit Language-Modellen wie in dieser Arbeit von Interesse, da hier zur Bewertung ja die Kontexte der Kanten essentiell sind. In diesem Abschnitt ist jedoch nur an die reine theoretische Schranke gedacht.

[5] Der Beam Decoder ist in Hauenstein & Weber 1994a und 1994b beschrieben ([39, 40]).

[6] Kita et al. 1989 [53] gehen in ihrem Ansatz einen Schritt in diese Richtung. Wortmodelle werden dabei grammatisch modelliert, ohne daß dabei Zyklen auftauchen. Die Phonmodelle, die in der Grammatik als terminale Symbole erscheinen, enthalten jedoch Zyklen zur Längennormalisierung. Bei Kita ist also das Aussprachelexikon des Decoders ein Teil der Grammatik des Parsers.

Wir vereinigen jetzt die Grammatik S, mit der ein Chartparser eine Lattice parsen würde, mit der Grammatik A, die die HMMs simuliert. Selbstverständlich müssen Variablen neu benannt werden, um Interferenzen zwischen S und A auszuschließen. Es ist klar, daß auch hier die Schranke von $O(n^3)$ für das vollständige Parsing gilt.

Aufgrund des *Local Ambiguity Packing* tauchen alle Worthypothesen, die von A erzeugt werden, in der resultierenden Chart genau einmal auf. Wenn wir nun die Chart bereits mit dem verbundenen Wortgraphen L_{max} initialisieren und dafür die Frames entfernen, kann die Komplexität des Parsing dadurch nicht ansteigen. Nur die Ableitungen, welche aus Frames Worthypothesen erzeugen, fallen weg.

Wenn wir nun desweiteren alle Regeln und Variablen von A wieder aus der Grammatik entfernen und nur wieder S zum Parsing verwenden, kann ebenfalls die Komplexität nicht steigen. Zudem bleibt das Parsingresultat gleich, da ja aufgrund der Umbenennung nur die Regeln von S zur Konstruktion von Kanten oberhalb der Wortebene verwendet werden konnten.

Wenn wir von L_{max} Worthypothesen entfernen, kann hierdurch ebenfalls die Komplexität nicht steigen.

Jede Lattice, die durch einen Decoder erzeugt werden kann, ist in L_{max} enthalten. Jede *echte* Lattice kann also aus L_{max} durch Entfernen von Worthypothesen erzeugt werden.

Daraus folgt, daß jede Lattice in $O(n^3)$ geparst werden kann, wobei n die Anzahl der Frames ist, die bei der Konstruktion der Lattice verwendet wurden.

Dem Leser, der glaubt, die Komplexität hänge vor allem von der Anzahl der Worthypothesen ab, die zu verarbeiten sind, sei gesagt, daß er recht hat. Aufgrund der Frames und der Lexikongröße ist jedoch eine echte obere Schranke für die Anzahl der Worthypothesen gegeben, die überhaupt entstehen können.

3.1.2.2 Obere Schranke abhängig von Wortgrenzenhypothesen

Eine einfache Überlegung zeigt bereits, daß die oben angegebene Schranke zu hoch ist. Eine Liste von Wörtern, wie beim "Standardparsing" ist ein Spezialfall einer Lattice. Offensichtlich ist aber die Anzahl der Wörter immer kleiner als die Anzahl der Frames, die zu deren Erkennung verwendet wurden. Darüber hinaus könnte eine gegebene Lattice L, die durch einen Decoder D_1 über einer Eingabe von n Frames erzeugt wurde, durch einen anderen Decoder D_2 über einer Eingabe von $2n$ Frames erzeugt werden. Offensichtlich gibt es keinen Grund anzunehmen, daß L, erzeugt durch D_2, ein schwereres Parsingproblem darstellt als L, erzeugt durch D_1. Da ein Decoder in der Praxis nicht über jedem Frame ein Worthypothesenende ausgibt, ist die Anzahl der tatsächlichen Frames ein zu großer Wert für n.

Jede Lattice verwendet von n Frames, die ein Decoder als Eingabe hatte, m Frames als Anfangs- und Endpunkte von Worthypothesen, wobei $m \leq n$ gilt. Die verwendeten m Frames nennen wir *Wortgrenzenhypothesen* einer Lattice ($WG(L)$).

Die Anzahl der Frames kann nicht kleiner sein, als die Anzahl der $WG(L)$, da eine Worthypothesengrenze immer auf einen Frame beim Decoding bezogen ist.

Für eine gegebenen Lattice L können wir also sagen, wieviele Frames minimal nötig waren, um L zu erzeugen, nämlich die Anzahl von $WG(L)$.

Da also das Parsing einer Lattice beschränkt ist durch $O(n^3)$, wobei n die Anzahl der Frames ist, und für jede Lattice L die Anzahl der Frames nicht größer sein muß als die Anzahl von $WG(L)$, ist die obere Schranke für das Parsing einer Lattice durch $WG(L)$ gebunden.

Definition 3.1 *Obere Schranke für Lattice-Parsing, wobei $WG(L)$, die in der Lattice L verwendeten Anfangs- und Endpunke von Worthypothesen sind.*

$$O(|WG(L)|^3) \tag{3.1}$$

Diese Schranke gilt immer noch, wenn ein inkrementeller Time Map verwendet wird, wie in Abschnitt 4.6.4. Die Operationen des Parsers sind dann nur nicht mehr in jedem Falle grammatische Operationen, sondern können auch Vererbungsschritte sein, wie beim Algorithmus 4.14 in Abschnitt 4.6.4.2.

3.1.3 Komplexität in der Praxis

Oben wurde eine theoretische obere Schranke für das Parsing von Lattices bestimmt. Bezogen auf die Praxis von Parsing von gesprochener Sprache folgen daraus einige Vorgehensweisen direkt:

- Es ist sinnvoll, Zeitabbildungen, wie etwa in Chien et al. 1990 vorgeschlagen, vorzunehmen. Die Reduktion von Knoten durch Zusammenfassung von Worthypothesen in der Chart verringert einen Parameter, der sich (wenigstens) kubisch auf das Zeitverhalten auswirkt.

- Sofern Grammatiken zum Parsing verwendet werden, die Ableitungseigenschaften von kontextfreien Grammatiken aufweisen, ist eine direkte Verarbeitung von Lattices bzw. Wortgraphen in angemessener Komplexität erreichbar. Theoretisch gilt die Schranke von $O(n^3)$ zwar nicht mehr, falls z.B. PATR-artige Grammatiken verwendet werden[7]. Die kritischen Fälle, die Parsing mit Unifikationsgrammatiken sogar unentscheidbar machen können, kommen jedoch in linguistischen Grammatiken nicht vor[8].

Insbesondere beim Schritt zum integrierten Speech-Language-Parsing mit Unifikationsgrammatiken, der in dieser Arbeit unternommen wird, hat der letztere Punkt beachtliche Auswirkungen. Kubisches Zeitverhalten ist nur zu erreichen, wenn die Möglichkeit besteht, *Structure Sharing* und *Local Ambiguity Packing* während des Parsing zu realisieren. Es folgen also unmittelbare Anforderungen an den Typ der einzusetzenden Unifikationsgrammatik. Eine HPSG-Grammatik, bei deren Parsing echte Relationen ausgewertet werden müssen, kommt z.B. nicht in Frage, um Wortgraphen direkt zu parsen. Die Subsumption ist für eine solche Grammatik nicht entscheidbar. Daher kann ein Parser keine gleichzubehandelnden Mehrfachparses zusammenfassen, wie es bei *Local Ambiguity Packing* notwendig ist. Dies würde immer zu einem exponentiellen Verhalten des Parsers führen.

[7] PATR-Parsing ist NP-vollständig bereits bei einer eindeutigen Eingabesequenz. Vergl. Perreira & Warren [68]

[8] Es handelt sich z.B. um unäre Produktionen, die immer wieder aus einer passiven Kante über einem Eingabeabschnitt neue Kanten erzeugen, die weder durch bereits bestehende subsumiert werden, noch selbst eine von diesen subsumieren. Eine einfache unäre Regel, die eine Liste akkumuliert, kann diesen Zweck bereits erfüllen. Eine Unifikationsgrammatik erlaubt unendlich viele sich nicht gegenseitig subsumierende Nonterminale in einer Ableitung . Ein solcher Fall ist jedoch ein "BUG" in der Grammatik.

Ein anderer wichtiger Punkt ist die Abhängigkeit der Verarbeitungskomplexität von der Mehrdeutigkeit bzw. der Konkurrenz von Hypothesen in einer Lattice. Theoretisch ist in L_{max} die Mehrdeutigkeit maximal. Sie ist im schlechtesten Fall (dem letzten Frame) $n \times k$, wobei k die Anzahl der Wortformen im Lexikon ist; in einem Knoten kann jede Wortform enden, mit jedem Vorgängerknoten als Beginn. Offensichtlich wird bei der oberen Schranke dieser Faktor mit zunehmendem n irrelevant. Dennoch ist die absolute Zahl der Worthypothesen, die eine Lattice enthält, ein gutes empirisches Maß für das Parsingproblem. Ein Beam-Decoder, wie der für die Experimente in dieser Arbeit verwendete, findet oft für ein Frame keine einzige Wortendehypothese, aufgrund seines Beams[9]. In der Praxis kann mit der Weite des Beams geregelt werden, wieviele Worthypothesen der Decoder erzeugt. Im Verhältnis zur Enge des Beams steigt die Zahl der Frames ohne Wortendehypothesen. Dies verkleinert das n, welches dominierend in die Parsingkomplexität eingeht. Als dritter Punkt, neben den oben erwähnten, kommt also hinzu:

- Es ist sinnvoll, durch zusätzliches Wissen die Menge an Worthypothesen früh zu beschränken.

Hinzu kommt, daß bereits bei einer Basiskomplexität von $O(n^3)$, die ja mit Unifikationsgrammatiken bei vollständiger Suche nicht wirklich erreicht werden kann, eine komplette Exploration des Suchraums prohibitiv ist. Der vierte wichtige Punkt ist also die Verwendung von Bewertungen, um eine unvollständige Suche[10] implementieren zu können, wie in Abschnitt 3.2.6. In diese Arbeit wird daher konsequent jeder Schritt bei der Verarbeitung durch statistische Modelle bewertet. Da im Verhältnis zu teuren Unifikationsoperationen die numerischen Operationen zur Berechnung der Güte von Schritten mit statistischen Modellen billig sind, ergibt sich also die folgende Konsequenz :

- Verwende billige statistische Restriktion vor teurer symbolischer Restriktion.

Obwohl wir nicht behaupten wollen, daß die hier entwickelte Vorgehensweise die einzig mögliche ist, werden die vier skizzierten Punkte in dieser Arbeit konsequent umgesetzt.

3.2 LR-Inkrementelle Chartanalyse und N-Gramm Statistik

"Inkrementell", wie Wirén[11] betont, kann in zweierlei Weise verstanden werden. Er unterscheidet *Full Incremental* vs. *LR-Incremental*. Während das erstere bedeutet, daß ein Parser in einer interaktiven Benutzerumgebung Eingaben des Benutzers (also z.B. auch Undo's) sofort schritthaltend verarbeitet, meint daß letztere die schritthaltende Verarbeitung mit der Sprechrichtung. Letzteres läßt sich als Spezialfall des

[9] Das Maximum, aufgrund dessen die Beam-Schwelle berechnet wird, liegt in einem solchen (häufigen) Fall in der Mitte einer Wortform.

[10] Mit "unvollständiger" Suche ist in dieser Arbeit eine Suche gemeint, die nicht den gesamten Suchraum explorieren kann. Eine Strahlensuche ist in diesem Sinne also unvollständig, während eine A*-Suche vollständig ist.

[11] [96]

ersteren auffassen. Wir sind hier ausschließlich an LR-Inkrementalität interessiert und wenden diese auf Lattices an. Der Sinn ist eine parallele Kopplung mit einem Beam-Decoder, der ebenfalls schritthaltend zeitsynchron — also LR-inkrementell — arbeitet.

In diesem Abschnitt wird der frameweise LR-inkrementelle ACP beschrieben. Die Unifikationsgrammatik, die in allen Experimenten benutzt wurde, wird vorgestellt und einige ihrer Eigenschaften motiviert. Darüber hinaus wird auf die (Un-)Wirksamkeit der "klassischen" Chartparsingtechniken beim Latticeparsing eingegangen, wie etwa *Top-Down Filtering* oder diverse Lookahead–Techniken.

Schließlich wird die Metrik, mit der statistische Bewertungen berechnet werden, und der verwendete Pruning-Mechanismus dargestellt und diskutiert. Dabei ist die Metrik so gestaltet, daß weitere Quellen statistischer Information leicht integriert werden können, wie z.B. in Kapitel 5 die direkte Integration der Unifikationsgrammatik als statistischem Modell.

3.2.1 Zeitsynchroner Chartaufbau

Das Parsingverfahren von Earley 1970 [24] baut seine Itemlisten von links nach rechts auf. Der ACP ist insofern eine Generalisierung des Earley Verfahrens, als daß die Reihenfolge, in der Itemlisten erzeugt und gefüllt werden, parametrisierbar ist. Bei der zeitsynchron inkrementellen Chartanalyse wird wieder ein Schritt zurück getan in Richtung von Earley. Die Parsingstrategie bleibt jedoch parametrisierbar, wie beim nicht-inkrementellen ACP. Die initiale Chart besteht nur aus dem Knoten 0. Die Chart wird erst verlängert, wenn neue Worthypothesen eingelesen werden. Jeder Knoten der Chart ist einem Zeitpunkt zugeordnet, mit festen Intervallen zwischen den Knoten. Ein gegenüber dem Worthypothesen generierenden Modul flexibler Ansatz kann wie folgt aussehen:

1. Erzeuge Knoten 0. Setze $T_{aktuell}$ auf 0
2. Lies neue Hypothese w_i Falls $T_{aktuell} < \text{end}(w_i)$, fülle die Chart mit Knoten von $T_{aktuell}$ bis $\text{end}(w_i)$. Setze $T_{aktuell}$ auf $\text{end}(w_i)$
3. Parse, bis Agenda leer ist.
4. Goto 2

Das links-rechts-Verhalten des Parsers mit der oben angegebenen Kontrollschleife variiert, je nachdem in welcher Reihenfolge die zu verarbeitenden Worthypothesen vorliegen. Wenn die Worthypothesen nach den Endzeitpunkten sortiert vom Parser eingelesen werden, ist der Effekt dieses Chartaufbaus eine extreme Breitensuche. Von links nach rechts gesehen werden alle Pfade in einem "Einlese- und Parse-Zyklus" quasi gleichzeitig verfolgt. Bemerkenswert ist, daß auch der Effekt einer sortierten Agenda völlig wegfällt. Lediglich innerhalb eines Items ergibt sich ein Best-First-Effekt. Falls die Worthypothese mit dem größten Endezeitpunkt zuerst beim Parser ankommt, wird sofort die ganze Chart initialisiert. Es resultiert eine Bestensuche über der ganzen Chart. Ob diese Bestensuche im Resultat einer Tiefensuche oder Breitensuche entspricht, hängt von der verwendeten Metrik für Kanten und Paare von Kanten auf der Agenda ab. In Abschnitt 3.2.6 wird hierauf näher eingegangen.

Beim Parsing von Lattices soll nicht der ganze Suchraum durchquert werden. Das Weglassen schlecht bewerteter Pfade (Pruning) ist unumgänglich im Hinblick auf ein Fernziel der Echtzeitverarbeitung. Um bei einer zeitsynchronen Verarbeitung den Suchraum nicht vollständig zu durchlaufen (Pruning), simulieren wir mit dem ACP das Verhalten eines Beam Decoders. Das heißt, wir verlängern die Chart immer nur genau um einen Knoten, und lesen dann alle Worthypothesen ein, die im zugehörigen Frame enden. Die Pfade mit zwei verschiedenen eingelesenen Worthypothesen in einem Zyklus konkurrieren. Diese müssen also beim Pruning verglichen werden. Das Verhalten des Chartparsers ist somit strikt links-rechts orientiert.

Die folgende vereinfachte Version einer Kontrollschleife, die in jeweils modifizierter Form in den Experimenten in 4.5 verwendet wurde, erzwingt dieses Verhalten:

1. Erzeuge Knoten 0. Setze T_{aktuell} auf 1
2. Lies alle neuen Hypothesen w_i, mit $\text{end}(w_i) = T_{\text{aktuell}}$
3. Prune
4. Parse, bis Agenda leer ist.
5. Inkrementiere T_{aktuell}, Goto 2

Diese Vorgehensweise läßt die Details sowohl des Pruning als auch des Parsing selbst völlig offen. Insofern ist auch noch nichts ausgesagt über die Art der verwendeten Unifikationsgrammatik und die Natur des statistischen Modells. Auch ist noch keine Beschränkung gegeben bezüglich der Art von Lattice, die durch die Menge von gelesenen Worthypothesen gebildet wird. Während der Experimente mit einer Online–Kopplung des LR-ACP und eines Beam-Decoders wird sich zeigen, daß die Menge der dabei übertragenen Hypothesen einen "von links verbundenen" Wortgraphen bilden. Dies ist jedoch keine Restriktion, die von der LR-Eigenschaft des Parsers herrührt oder nur ein Merkmal des Beam-Decoders. Es handelt sich um eine Eigenschaft der zeitsynchronen Kopplung von Parser und Decoder.[12].

3.2.2 Verwendete Unifikationsgrammatik

Wie bereits oben angedeutet wurde, ist nicht jede Variante von Unifikationsgrammatiken bzw. Formalismen für eine effiziente Verarbeitung von Lattices geeignet. Insbesondere gilt dies für eine zeitsynchrone Verarbeitung. Die Grammatik, die hier verwendet wird, ist so gestaltet, daß sie den folgenden Anforderungen genügt:

- Ein Subsumptionstest auf zwei Merkmalsstrukturen ist grundsätzlich möglich. Wie bereits angedeutet wurde, ist dies nötig, um effiziente Parsingschemata beibehalten zu können. Immer, wenn in der Chart eine passive Kante eingetragen wird, kann dann überprüft werden, ob bereits eine Kante existiert, deren Merkmalsstruktur die der ersten subsumiert, oder von ihr subsumiert wird.

[12] Siehe auch Abschnitt 4 für eine ausführliche Diskussion von linksverbundenen Wortgraphen.

Nur die speziellere bzw. allgemeinere[13] der beiden wird dann weiterverfolgt, um *Local Ambiguity Packing* zu simulieren[14].

- Die Regeln der Grammatik sind im Stile von PATR-Regeln geschrieben. Das heißt, man kann eine Regel trennen in einen kontextfreien Regelanteil und eine Menge von Restriktionen, die als Merkmalsstrukturen kodiert sind. Dies hat zweierlei Gründe:
  - Effizientes top down Parsing bleibt auch mit Unifikationsgrammatik möglich. Beim Prädiktorschritt wird dann nur die Information benutzt, die im kontextfreien Teil der Regeln kodiert ist. Ein "Restriktions"-Mechanismus, wie bei Shieber 1985, kann hierdurch billig simuliert werden.
  - Die Ableitungen der Unifikationsgrammatik können unkompliziert als statistisches Modell benutzt werden. Wie in Kap. 5 dargestellt, kann die Unifikationsgrammatik dadurch mit Varianten von Trainingsverfahren für kontextfreie Grammatiken trainiert werden.
- Restriktionen sind beschränkt auf Regeln. Dadurch ist beim Parsing garantiert, daß jeweils lokal entschieden werden kann, ob die Kombination zweier Kanten fehlschlägt, oder zu einer neuen Kante führt. Dies erlaubt durchaus die Einbettung von Funktionen in den Formalismus, die sofort ausgewertet werden können. Volle Relationen jedoch, wie sie üblicherweise in HPSG verwendet werden, scheiden aus. Dasselbe gilt für ID-LP-Grammatiken generell, deren LP-Bedingungen global sind. Diese Eigenschaft wird essentiell wichtig, wenn vom Parser lokal entschieden werden soll, welche Wörter einen partiellen Parse verlängern könnten. Dies geschieht z.B. in Kapitel 4.4, wo der Decoder von links nach rechts nur vom Parser vorgeschlagene Wortformen mit der Signaldarstellung vergleicht. Je weniger Lokalität die Grammatik erlaubt, desto unsicherer werden die Vorhersagen des Parsers.

Der Formalismus, in dem die verwendete Unifikationsgrammatik geschrieben wurde, ist detailliert in Euler 1992 dargestellt[15] [29, 30]. Eine Merkmalsstruktur besteht aus einer Konjunktion von Merkmalen und deren Werten und einem Typ. Werte sind wieder Merkmalsstrukturen, bzw. Disjunktionen von Merkmalsstrukturen, oder atomare Typen.

Ein Beispiel einer Grammatikregel ist die "PP-RULE" in Abbildung 3.2, die Präpositionalphrasen aus Präpositionen und deren Argumenten aufbaut. Der Wert des `sem`-Attributs transportiert die Beschreibung der Semantik. Wie in HPSG wird über ein `subcat`-Attribut, in das über Koreferenz das Argument instanziiert wird, die spezielle semantische Spezifikation transportiert.

[13] Beide Strategien sind möglich. Falls die allgemeinere Merkmalsstruktur verfolgt wird, steht bei einem Fehlschlag der Fehlschlag der spezielleren ebenfalls fest und muß nicht mehr geprüft werden. Falls die speziellere Merkmalsstruktur verfolgt wird, steht bei einem erfolgreichen Parse fest, daß die allgemeinere auch erfolgreich ist, und dies muß ebenfalls nicht mehr geprüft werden. Verbreiteter ist die erste Strategie.

[14] vergleiche hierzu im Besonderen Shieber 1985 [86].

[15] Ich danke ausdrücklich Lutz Euler (damals Universität Hamburg, NatS), der bei der Entwicklung des ASL-Formalimus auf zahlreiche meiner Sonderwünsche eingegangen ist.

```
rule[(name "PP-RULE")
     (rule <
            pp[(syn [(subcat nil)
                     (wh %3=[])
                     (ptype %2)])
               (sem %10)]

             p[(syn [(subcat <%1>)
                     (ptype %2=[])])
               (sem %10=[])]

       %1=parg[(syn [(wh %3)])]
                                         >)]
```

Abbildung 3.2: Eine Grammatikregel, die Präpositionalphrasen aus Präpositionen und deren Argumenten aufbaut.

```
p[(phon "NACH")
  (syn [(ptype to)
        (subcat < parg [(syn [(case akk)
                              (xtype local)])
                        (sem [(content %4=[(var %2=[])])
                              (last nil)])]>)])
  (sem [(content [(var %11=[(sorte [(dynamic yes)])])
                  (cond [(pred goal)
                         (theme %11)
                         (goal %2)])])
        (quant [(list < %4 . %5 >)
                (last %5=[])])])]
```

Abbildung 3.3: Der Lexikoneintrag für lokales *nach*.

Die ganze Regel ist vom Typ `rule`. Das Merkmal `rule` hat als Wert eine Liste von Merkmalsstrukturen. Die Liste wird von Parser als kontextfreies Regelschema interpretiert. Abbildung 3.3 zeigt einen Lexikoneintrag für die Präposition *nach*[16].

Die linguistischen Details der Regeln sind für diese Arbeit weniger von Bedeutung. Wichtig ist jedoch, daß die Grammatik eine Semantik im Stile von HPSG aufbauen kann. Obwohl dies ein noch relativ unerforschtes Gebiet darstellt, verbindet sich damit die Hoffnung, mächtigere Grammatiken auf ein solches Format auskompilieren zu können. In der verwendeten Unifikationsgrammatik wurden die folgenden Mittel verwendet:

- Konjunktion von Merkmalen
- Disjunktion von Merkmalen
- Koreferenz von Pfaden.
- Schwache Typen

Auf volle Relationen und Negation ist verzichtet worden, da hier eine effiziente Verarbeitung, wie sie für Lattice-Parsing nötig ist, mit derzeitigen Mitteln nicht erreichbar scheint.

3.2.3 Anwendung von klassischen Strategien

Im Vergleich zu kontextfreien Grammatiken ist die Grundvergleichsoperation bei Unifikationsgrammatiken sehr teuer. Um eine angemessene Effizienz zu erreichen, die z.B. die Durchführung von zahlreichen Experimenten oder ein Training der Grammatik überhaupt erst erlaubt, mußten zahlreiche Mechanismen verwendet und zum Teil erst entwickelt werden. Bei einem Vokabular von 500 Wörtern mußten bei den durchgeführten Experimenten zwischen 500 und 700 Worthypothesen erzeugt werden, um für eine Äußerung der Länge von 5 Sekunden alle gesprochenen Wörter in der Lattice zu erhalten. Dabei wurden etwa 200 verschiedene Anfangs- und Endpunkte verwendet. Dies ist auch für polynomiale Verfahren ein großes Problem, im besonderen wenn die Grundoperation bereits teuer ist.

In der Fachliteratur sind bereits zahlreiche Modifikationen der in Abschnitt 2.3.2 vorgestellten bottom up und top down Parsingstragien diskutiert worden – immer im Kontext der Verarbeitung einer deterministischen Eingabe. Bei Wirén [96] findet sich ein guter Überblick. Die folgenden Abschnitte diskutieren diese Techniken im Hinblick auf ihre Verwendbarkeit für das Parsing von Lattices und hierbei im besonderen im Hinblick auf LR-Inkrementalität. Es zeigt sich, daß einige der Techniken dabei völlig wirkungslos werden, andere sich sogar negativ auswirken und mit dem Konzept des LR-inkrementellen Latticeparsing nicht verträglich sind.

3.2.3.1 Bottom-Up Filtering

Bottom-Up Filtering bezeichnet die bereits von Earley bekannte Technik des Lookahead bezüglich terminaler Symbole einer Grammatik. Bereits bei Ney 1991 [65]

[16] Die Einträge für die Semantikspezifikation, sowohl der verwendeten Regeln als auch der Lexikoneinträge der Unifikationsgrammatik stammen von Walther Kasper, DFKI Saarbrücken.

findet sich die Beobachtung, daß sich beim Parsing von Lattices verschiedene chartbasierte Verfahren für CFGs mit und ohne Lookahead gleich verhalten[17]. Dies gilt nicht nur für Parsing mit CFGs. Es handelt sich um eine Eigenschaft von Latticeparsing. Zumeist wird der Begriff *Bottom-Up Filtering* darauf bezogen, daß die durch den Lookahead gegebene Restriktion bei top down Kanteneinführung durch einen rekursiven Prädiktor-Schritt die Menge eingeführter Kanten beschränkt. Auch andere Schritte sind so beschränkbar.

Für eine Lattice ist bereits intuitiv klar, daß die Beschränkung durch zu scannende Terminale zur rechten schwächer sein muß als bei deterministischen Eingaben. In jedem Punkt der Chart können nämlich viele Worthypothesen beginnen. In der Praxis kommt der Fall extrem selten vor, daß nur eine Worthypothese zur rechten vorliegt.

Falls beim Prädiktorschritt (top down) kein Bottom-Up Filtering verwendet wird, dann ist, nachdem die Einführung der Kanten abgeschlossen ist, eine Menge $Term_{aktuell}$ von Terminalen durch die Kanten erreichbar. Offensichtlich ist $Term_{aktuell}$ eine Teilmenge des Lexikoninventars. Wenn wir nun L_{max} betrachten (als schlechtesten Fall) so beginnt in jedem Chartknoten jedes Wort des Lexikoninventars. Es ergibt sich also keine Einschränkung. Natürlich kommt in der Praxis L_{max} nicht vor.[18] Dennoch ist das Argument valide. Denn bereits eine Teilmenge des Lexikons ($Term_{aktuell}$) reicht aus, damit ein Lookahead keine Beschränkung auf den Kanten mehr zuläßt. Die grundsätzliche Beobachtung ist, daß mit dem Verzweigungsgrad der zu parsenden Lattice der Grad der Beschränkung sinken muß, während der Rechenaufwand für den Lookahead steigt.

Obwohl Erreichbarkeitstabellen offline gut auskompiliert werden können, reicht das erste Argument aus, um für Latticeparsing ein *Bottom-Up Filtering* als wirkungslos zu verwerfen.

Wirén gibt empirische Ergebnisse an über Effizienzvergleiche mit verschiedenen Kombinationen der hier diskutierten Techniken. Diese Untersuchungen wurden zwar nur auf deterministischen Eingaben durchgeführt. Dennoch sagen sie etwas aus über die Kraft der Beschränkung, die durch ein Regeleinführungsschema entsteht. Dabei zeigte sich, daß *Bottom-Up Filtering* nur in Kombination mit dem unten diskutierten *Kilbury Schema* signifikante Verbesserungen erzielt.

Die letztere Technik ist jedoch im Zusammenhang mit Latticeparsing ebenfalls uninteressant, wie in Abschnitt 3.2.3.3 begründet wird.

Die bisherigen Argumente gelten für die Verarbeitung von Lattices im Allgemeinen. Bezüglich einer LR-inkrementellen Verarbeitung des ganzen Systems, also Worthypothesenkonsument (Parser) und Worthypothesenproduzent (Decoder) ist ein Lookahead bezüglich terminaler Symbole grundsätzlich nicht anwendbar, weil zum Zeitpunkt der Regeleinführung die benötigten Terminale noch nicht vorliegen.

3.2.3.2 Top-Down Filtering

Top-Down Filtering ist die komplementäre Technik zu *Bottom-Up Filtering*. Während bei BUF top down Regeleinführungen bottom up beschränkt werden, verhält es sich bei TDF exakt anders herum. Vorausgesetzt, aktive Kanten werden bottom up aufgrund einer passiven Kante eingeführt, dann läßt sich die Menge der

[17] Ney vergleicht CYK [100] und Earley [24].

[18] In unseren Experimenten treten immer noch Verzweigungen von 20 auf.

so eingeführten Regelkanten beschränken, indem überprüft wird, ob es möglich ist, jemals eine bereits im diesem Knoten endende aktive Kante zu erreichen.

Grundsätzlich gilt für TDF dasselbe Argument wie für BUF: Beim Parsen einer Lattice enden in Vorgängerknoten viele Worthypothesen. Demzufolge sind entsprechend mehr konkurrierende Analysen möglich, was sich in den aktiven Kanten äußert, die im Vorgängerknoten enden. Die Beschränkung auf bottom up einführbare Kanten ist entspechend geringer.

TDF ist jedoch mit LR-inkrementellem Parsing verträglich. Zum Zeitpunkt des Einlesens neuer Worthypothesen ist die linke Portion der Eingabe bis zu $T_{aktuell}$ ja bereits geparst. Die nötige Information für das Filtern steht also zur Verfügung. TDF scheidet also daher als Kandidat für eine — wenn auch geringe — Effizienzsteigerung nicht a priori aus.

3.2.3.3 Kilbury-Schema

Die *Kilbury Strategie* ist eine modifizierte Variante der klassischen bottom up Kanteneinführung, wie in 2.3.2 dargestellt. Bei KS wird jedoch keine leere neue Kante eingeführt, sondern der Parsingschritt über passive Kanten hinweg wird sofort ausgeführt. Außer der Startkante existieren leere aktive Kanten nicht bei KS.

Bei einem Vergleich verschiedener Strategien in Wirén 1992 [96] nur mit deterministischen Eingaben, schneidet KS mit BUF und TDF am besten ab. Dieses Ergebnis ist jedoch nicht auf Latticeparsing übertragbar.

Der Erfolg des Kilbury Verfahrens beruht für deterministische Eingaben darauf, daß ein Schritt immer sofort ausgeführt wird, wenn die nötige Information zu Verfügung steht. Dieser Schritt muß sonst zu einem späteren Zeitpunkt sowieso ausgeführt werden, wobei dieselbe Information nochmals berechnet wird. Die KS spart somit für jede bottom up Kanteneinführung eine Kante und eine zusätzliche Vergleichsoperation.

Für Latticeparsing mit Unifikationsgrammatiken gilt jedoch generell, daß der Suchraum nicht vollständig durchquert werden kann. Das Kilbury Verfahren führt also eventuell Schritte sofort aus, die ansonsten nie ausgeführt werden, da sie beim Pruning-Schritt wegfallen. KS wiederspricht insofern dem Paradigma des statistisch getriebenen Parsing, nämlich *billige vor teurer Restriktion*.

Während sich bei einem statistisch getriebenen Parsing über einer bereits voll initialisierten Chart diese Strategie nachteilig auswirken kann, ist sie mit LR-inkrementellem statistisch getriebenen Parsing, wie es in den folgenden Kapiteln dargestellt wird, gänzlich unvereinbar. Es wird sich zeigen, daß während eines Zyklus der Kontrollschleife wie in 3.2.1 der Zeitpunkt, nachdem leere Kanten eingeführt wurden, am besten geeignet ist, um den Pruning-Schritt auszuführen. Genau dieser Verarbeitungszeitpunkt wird bei KS übersprungen.

3.2.3.4 Top-Down mit Shiebers *Restriction*

Die bisher diskutierten Techniken betreffen das Verarbeitungsschema des Aktiven Chartparsers, unabhängig von der Art der verwendeten Grammatik. Der unter dem Namen *Restriction* bekannt gewordene, auf eine Idee von Shieber 85 [86] zurückgehende Mechanismus ist direkt auf die Verarbeitung von Unifikationsgrammatiken bezogen. Ein Debatte über diverse Anwendungen dieses Mechanismus in ACP findet sich bei Gerdemann 1989 [35].

Diverse Operationen über Unifikationsregeln, wie etwa die top down Kanteneinführung der SEEK-DOWN Operation können in Endlosschlaufen laufen, wenn die volle Unifikation aller Merkmale verwendet wird. Daher schlägt Shieber 1985 vor, in solchen Fällen nur einen festen Teil der Merkmale und Werte (eben die *Restriction*) für die Unifikation zu benutzen, die garantiert zu endlich vielen Ausprägungen führt.

Gerdemann 1989 [35] berichtet, daß mit dieser Technik die Parsingzeit im Experiment vom Minutenbereich in den Sekundenbereich gesenkt werden konnte. Dies entspricht unseren Beobachtungen. Dennoch erscheint der Schritt, den Shieber vorschlägt, noch nicht konsequent zuende gedacht. Obwohl die Verwendung der *Restriction* voll mit Latticeparsing verträglich ist, wie auch mit zeitsynchronem Vorgehen, reicht der Effizienzgewinn, zumindest in unserer Testimplementation, nicht aus, um den Schritt von der deterministischen Eingabe zur Wortlattice mit angemessenen Parsingzeiten vollziehen zu können.

Unabhängig von der Parsingstrategie im ACP[19] müssen auch bei der Verwendung von Shiebers Mechanismus' noch viele Operationen ausgeführt werden, um eine neue aktive Kante einzuführen:

- Für jedes Regelobjekt muß die *Restriction* gebildet werden.
- Eine Unifikation der *Restriction* mit einem Teilgraph der Regel muß für jede Regel ausgeführt werden.
- Ein Subsumptionstest, zumindest der *Restrictions* zweier Regeln muß für jede neue Regel mit jeder bereits eingeführten vorgenommen werden.[20]

Daher haben wir eine effiziente Weiterentwicklung der *Restriction*-Idee vorgenommen, die den Aufwand für eine top down Kanteneinführung auf eine kleine Konstante reduziert. Diese Technik, die im folgenden Abschnitt beschrieben wird, macht auch die Filter-Techniken obsolet. Die vollständige Vorverarbeitung der SEEK-DOWN Operation wäre nur mit einem Bottom-Up Filtering noch zu verbessern. Dieses ist jedoch — wie bereits betont wurde — nur für die Verarbeitung von deterministischen Eingaben geeignet.

3.2.4 Regelvorverarbeitung im LR-ACP

Die Unifikation ist eine so teure Operation, daß es nötig ist, soviele Schritte wie möglich vorzuverarbeiten, um zu einem effizienten Verarbeitungsmodell zu gelangen.

Im LR-ACP werden daher nur noch die Unifikationen der Completeroperation während der Laufzeit ausgeführt. Die Scanner-Operation verwendet keine Unifikationen bei Verwendung eines Vollformenlexikons und die SEEK-DOWN Operation wird im hier beschriebenen Ansatz vollständig vorverarbeitet. Dieser Vorverarbeitung ist der Abschnitt 3.2.4 gewidmet.

3.2.4.1 Simulation von Shiebers *Restriction* durch Typen

Obwohl die *Restriction* für eine gegebene Merkmalsstruktur sehr klein sein kann (sie wird definiert), muß für deren Verarbeitung immer noch der Unifikator bemüht wer-

[19] TD oder BU.

[20] Siehe auch Gerdemann 1989 [35].

den. Falls dieser destruktiv ist, erfordert die Unifikation teures Kopieren, falls nicht, immmer noch das Verwalten der Kontexte. Um dies zu umgehen, kodieren wir die relevante Information direkt in den Typ einer Merkmalsstruktur. Die Typunifikation allein (ohne die Merkmale und Werte der MS) ist $O(k)$[21] und im Formalismus effizient auf Bitvektorebene implementiert.[22] Das schwache Typenkonzept trennt die Typunifikation vollständig von der Merkmalsunfikation ab. Die Typen selbst haben endlich viele Ausprägungen und der Verband ist dadurch voll in Bitvektoren auskompilierbar. Anstatt der Unifikation mit der *Restriction* unifizieren wir nur die globalen Typen der Regeln. Dies hat die folgenden Vorteile:

- Die Information bezüglich der Auswahl einer *Restriction* ist fest kodiert und muß nicht errechnet werden.
- Typunifikation ist nicht destruktiv und ändert nichts an den Regelobjekten. Es besteht weder Kopier- noch Verwaltungsaufwand von Kontexten.
- Da die Regelobjekte nicht verändert werden, entfällt der Subsumptionstest beim Einführen leerer Kanten ganz. Alle Regelobjekte werden durchnumeriert und der Redundanztest bei der Einführung von Regeln besteht nur noch aus dem Vergleich zweier Zahlen.

Diese Abbildung auf endlich viele Instanzen von Regeltypen erlaubt weitere Schritte in Richtung von Kompilationstechniken, die im nächsten Abschnitt geklärt werden.

Die linguistische Fragestellung, was in einer Grammatik als Typ kodiert sein sollte, ist natürlich keineswegs gelöst. Sie fällt jetzt zusammen mit der Frage, welche Merkmale für eine *Restriction* in Frage kommen. Üblicherweise wird hier mit Begriffen wie “die empirisch maximal restriktive Information” hantiert. In der Praxis verwenden wir *Part-of-Speech-* und *Subcat*-Information.

Gerdemann 1989 weitet die Anwendung von Shiebers Mechanismus auf verschiedene Schritte beim ACP aus. Durch unser Typsystem weiten wir den Mechanismus auf alle Unifikationen aus. Jede Unifikation zweier MS erfolgt nach dem folgenden Schema:

1. Unifiziere die Typen der MS
2. Falls dies gelingt, kopiere und unifiziere die ganzen MS

Es handelt sich hier um eine Ausprägung des Prinzips “Billige Restriktion vor teurer Restriktion”[23]

3.2.4.2 Generelle Regelvorverarbeitung

In der Literatur gibt es eine Reihe von weiteren Mechanismen, wie Parsing effizient betrieben werden kann. Neben parallelisierenden Verfahren, wie etwa [4], die an dieser Stelle weiter keine Beachtung finden, sind Tabellenverfahren denkbar. Der in Kap. 2.3.1 beschriebene GLRP[24] nutzt zur effizienten Verarbeitung aus, daß

[21] k ist die Länge der Bitvektoren und hängt von der Menge der Typen ab.
[22] Siehe Euler 1992 [29, 30]. Der verwendete Unifikationsformalismus ist dort beschrieben.
[23] ...der auch das Leitparadigma des ganzen Abschnitts 5 bildet.
[24] Für weitere Literaturreferenzen siehe Abschnitt 2.3.1.

Regelfolgen im kontextfreien Fall sich vorher in eine Sprungtabelle auskompilieren lassen. Viele grammatische Tests werden dabei gespart.

Auch beim ACP lassen sich statische Regelfolgen vorher in Tabellenform ablegen, um Rechenaufwand zu sparen. Theoretisch gibt es kein Hindernis, eine Chart durch ein GLRP-Verfahren aufzubauen[25]. Staab 1994, [89] der eine Variante des hier vorgestellten LR-ACP mit einer kontextfreien Grammatik entwickelt, adaptiert unseren Agenda-Mechanismus zum Pruning, der in späteren Abschnitt 3.2.6.2 und 3.2.6.3 vorgestellt wird.

Eine Alternative ist die Vorherberechnung der SEEK-DOWN Operation, sodaß nur noch der Redundanztest beim Einführen neuer leerer Kanten anfällt.

Die hier beschriebenen Techniken sind bei Unifikationsgrammatiken im Stile von PATR anwendbar, sofern das Typsystem gewissen Ansprüchen genügt.[26].

Durch die Verwendung von globalen Typen der Regelinstanzen als kontextfreies Gerüst einerseits und die Simulation von Shiebers *Restriction Parsing* ergeben sich immer endlich viele Regelinstanzen. Wie bereits betont, ist ein Subsumptionstest dadurch beim top down Einführen von leeren Kanten nicht nötig. Der rekursive Aufruf der *SEEK-DOWN*-Operation terminiert, wenn in allen Verzweigungen des rekursiven Abstiegs keine neue Regel mehr eingeführt werden kann. Da durch die Typunifikation die Regelgebilde nicht verändert werden, kann diese Folge von Regeln für jeden möglichen Typ vorherberechnet werden. Sie ist statisch. Dies ist dann gewährleistet, wenn kein Typsystem mit rekursiver Typinferenz verwendet wird.

Beim Errechnen der Folge ist durch die rekursiven Aufrufe eine Reihenfolge festgelegt, in der die Typunifikationen gelingen. Indem wir nun mehr Information hinzufügen, nämlich die Unifikation der ganzen Merkmalsstrukturen, werden zusätzliche Folgen ausgefiltert.

Wir tun dies jedoch nur beim Errechnen der Regelfolgen. Eingefügt werden nur die Regelobjekte ohne die zusätzlichen *hineinunifizierten Merkmale und Werte.*

Wir können so also:

- Die gesamte SEEK-DOWN-Operation off-line berechnen und das Ergebnis in einer Tabelle halten, deren Schlüssel jeweils Typen sind und deren Einträge Regeln sind.

- Die maximale Beschränkung der gesamten Unifikationsoperation beim Einführen von leeren Regeln anwenden. Nur der allererste Schritt ist hiervon ausgenommen. Beim Anstoßen des SEEK-DOWN durch eine nicht leere aktive Kante wird nur der Typ betrachtet, der als Hash-Key verwendet wird. Dies ist nötig, um die Tabellenschlüssel endlich zu halten[27].

Diese Technik ist im übrigen bei bottom up Parsingstrategie nicht anwendbar. Hier können nur die unmittelbaren Regeleinsetzungen aufgrund von Terminalen fixiert werden. Alle anderen passiven Kanten, die nicht nur ein Terminal überdecken,

[25] Eine Agenda-getriebener GLRP über einer Chart für Worthypothesen wird derzeit in Staab [89] entwickelt. Es handelt sich hierbei um eine parallele Entwicklung zum LR-ACP.

[26] Der Anstoß, sich mit der Vorwegkalkulation von Regelfolgen zu befassen, kam von Günther Neumann, DFKI Saarbrücken, der mich 1991 in einem Gespräch auf diese Möglichkeit beim ACP hinwies. Er befaßte sich damals innerhalb des Projektes DISCO mit der Vorherberechnung von Regeleinführungsmechanismen in der Chart.

[27] In der Praxis sind es endlich viele Ausprägungen für einen Korpus. Das Argument gilt dennoch; man ersetze nur "endlich" durch "klein und schnell vergleichbar".

entstehen dynamisch und können theoretisch nicht auf endlich viele Instanzen beschränkt werden, ohne auf die volle Unifikation zu verzichten und nur das kontextfreie Rückrat der Typen zu verwenden. Damit aber wäre zum Beispiel gegenüber dem GLRP kein Vorteil gegeben. Die Anwendung der Technik bei bottom up Parsing, nur unter Verwendung der Typinformation entspricht in etwa der Verwendung eines nichtdeterministischen LR-Tableaus beim GLRP. In beiden Fällen müssen alle Merkmalsunifikationen während der Laufzeit ausgeführt werden, jedoch nur in den Fällen, in denen die Tabelle angibt, daß die Typunifikation gelingt[28]

Wir verwenden die top down Regeleinführung in allen Experimenten mit Lattices in dieser Arbeit. Mit allen Techniken, die hier vorgestellt werden, hat sie sich als verträglich erwiesen.[29]

3.2.5 Statistisches Language Modell

Um Lattices effizient parsen zu können, ist es — wie oben bereits bemerkt — nicht mehr möglich, den Suchraum vollständig zu durchqueren. Die Standardmethode, die Suche zu beschränken, ist die Verwendung von Bewertungen der Erkennungsgüte aus der Akustik. Bestensuche in der Chart, die diese Bewertungen verwendet, wurde von verschiedenen Autoren vorgestellt, u.a. von Päseler [67], Kogure 1989 [55] , Thompson 1990 [90], Ney 1991 [65], Chien et al. 1990 [16] und Weber 1992 [95][30].

Eines der Hauptargumente für eine Kopplung von Parsing und Decoding mit einer N-Beste-Schnittstelle, die bei verschiedenen Autoren wiederholt wurde ist etwa paraphrasiert wie folgt:[31]

- Wende billige Restriktion vor teurer Restriktion an.

Dieses bestechende Paradigma, dem aus der Sicht des Softwareengeneering kaum widersprochen werden kann, wenden wir nun konsequent innerhalb des ACP an. Der Unterschied zur N-Beste-Schnittstelle ist jedoch die Granularität der Operationen, bezüglich derer billige statistische vor teurer symbolischer Restriktion angewendet wird. Wir kombinieren die Suche und die Analyse, wobei vor jedem Analyseschritt die Güte des Schrittes vorher durch ein statistisches Modell bewertet wird.

Im ACP wird beim Latticeparsing, analog zur Verarbeitung deterministischer Eingaben, ein möglicher Suchschritt dargestellt durch ein Paar von aktiver und passiver Kante. Jedes dieser Paare überspannt einen Teilpfad von Worthypothesen in der Lattice. Wir verwenden also das statistische Languagemodell, um jedes Paar zu bewerten, bevor die grammatische Operation (die Fundamentale Regel des ACP) angewendet wird.

Chien et al. 1993 [17] schlagen ebenfalls ein ähnliches Verfahren vor[32], Bi-Gramm-Modell und Unifikationsgrammatik lose zu koppeln. Sie konzentrieren sich

[28] Der GLRP nutzt noch mehr Information aus, die im LR-Tableau kodiert ist. Um den BU-ACP wirklich mit der gleichen vorverarbeiteten Information auszustatten, müßten bei der Vorverarbeitung noch die Reihenfolgen der Anwendbarkeit leerer Kanten mitkodiert sein. Dies ist durchaus möglich.

[29] In diversen Tests über deterministischen Eingaben ergab diese Regelvorverarbeitung eine Halbierung der Zeit für das Parsen eines Satzes.

[30] Sicher ist diese Literaturangabe unvollständig. Es handelt sich jedoch, wie gesagt, um eine Standardvorgehensweise

[31] Siehe u.a. Chow & Schwartz 1989 [18] oder Schwartz & Austin 1990 [80].

[32] Im Frühjahr 1993, als die Vorarbeiten und ersten Experimente zu dem hier präsentierten Verfahren bereits stattgefunden hatten, erschien die Arbeit von Chien et al., die unabhängig auf

jedoch in ihrer Arbeit auf eine klassische Bestensuche über der ganzen Breite der Chart.

Chien et al. 1993 verwendeten ein wortformorientiertes Bi-Gramm-Modell, daß über einer Sammlung chinesischer Schulbücher trainiert wurde. Die Perplexität von 30.2 scheint sehr niedrig zu sein, für ein Modell das über Sprachdaten, die nicht domänenspezifisch sind, trainiert wurde. Möglicherweise sind im Chinesischen die Restriktionen bezüglich der Serialisierung rigider als im Deutschen.

Die Experimente in dieser Arbeit sind allesamt mit dem gleichen Bi-Gramm-Modell ausgeführt worden, das über Dialogen der Zugauskunftsdomäne trainiert wurde. Die Testsatzperplexität des Modells nach Jelinek et. al. 1983 [45] wurde mit 54.28 gemessen.

Eine Kopplung von Bi-Gramm mit unifikationsbasiertem Latticeparsing innerhalb eines GLRP findet sich auch bei Schmid 1994 [79]. Die Suche in der Lattice und das eigentliche Parsing sind hier jedoch vollständig voneinander abgekoppelt. Daher ist das Verfahren nicht für unsere Zwecke geeignet. Bei Schmid 1994 werden mithilfe von kombinierter akustischer und Bi-Gramm-Bewertung sukzessive Pfade aus der Lattice extrahiert und einem GLRP für deterministische Eingaben übergeben. Dieses Vorgehen entspricht einer Simulation einer N-Beste-Schnittstelle auf der Basis einer Wortgraphenschnittstelle, mit allen Vor- und Nachteilen, die in Abschnitt 2.3.3.1 beschrieben wurden. Schmid 1994 führt dabei eine gepackte Darstellung der N-Beste-Ketten Repräsentation ein. Das Packen erfolgt jedoch nur von links. Es entsteht also ein Baum, der mit einer A*-Suche durchlaufen wird. Dabei wird das Bi-Gramm-Modell im Grunde doch wieder aus dem eigentlichen Parsing herausgezogen.

In Staab *forthcoming* [89] wird eine engere Kopplung vogenommen, die in ihren Grundzügen den hier vorgeschlagenen Methoden entspricht. Die Verwendung einer passiven Chart im GLRP und eine Agenda-getriebene Kontrolle kommt dem im folgenden Abschnitt dargestellten Ansatz sehr nahe.[33]

Obwohl die Vorgehensweise von Schmid 1994 bei ausreichender Erkennungsgüte zu guten Resultaten führt, verfolgen wir diesen Ansatz nicht weiter, da er mit LR-Inkrementalität schwer vereinbar ist.

Die Metrik und der Pruning-Ansatz, der im nächsten Abschnitt beschrieben werden, sind mit jeder Art von N-Gramm-Modell verträglich.

Alle Experimente wurden nur über einem Vokabular von 500 Wörtern ausgeführt, da die Unifikationsgrammatik nicht mehr abdeckt. Für eine effiziente Implementierung des Prediction-Protokolls des ACP zur top down Beschränkung des Decoders ist ein klassenbasiertes Modell Voraussetzung[34]. Dies wird unten weiter diskutiert werden.

---

dieselbe Grundidee einer losen Kopplung von Bi-Gramm-Modell und Unifikationsgrammatik im Chartparser gestoßen waren.

[33] Durch die Verwendung der passiven Chart und einer Agenda zeigt Staab *forthcoming*, daß eine Strahlensuche über der Agenda mit kombinierten Bewertungen, wie sie hier entwickelt wird, auch mit einem GLRP verträglich ist.

[34] Ein einfaches Klassenmodell genügt. Es muß sich nicht um ein doppelt stochastisches Modell handeln, in der Art von Kneser und Ney 1991 [54], obwohl dies auch verträglich ist.

3.2.6 Metrik und Pruning

In diesem Abschnitt wird auf die wenigen bisherigen Ansätze eingegangen, die Information statistischer Sprachmodelle (Languagemodelle) in der Chartsuche verwenden. Es stellt sich dabei heraus, daß diese Ansätze teilweise unkorrekt oder zum gewünschten Zweck nicht einsetzbar sind. Eine eigene Metrik wird vorgestellt, die einige Probleme besser löst als die bisherigen Ansätze.

3.2.6.1 Bisherige Ansätze

Die Ansätze in der Literatur, die statistische Languagemodelle und Parsing über Wortgraphen eng verzahnen, sind nicht zahlreich. Neben Hauenstein & Weber 1994 [39, 40][35], die eine vereinfachte Form des hier vorgestellten Verfahrens verwenden, kombinieren Chien et al. 1993 aktives Chartparsing über Lattices und Suche mit einem Bi-Gramm-Modell.

Der Ansatz von Chien et al. 1993 ist agendabasiert. Jedes Paar von Kanten, das dem ACP-Algorithmus zufolge auf die Agenda gestoßen wird, erhält eine Bewertung. Als Bewertung wird direkt die Bi-Gramm-Wahrscheinlichkeit des vom Kantenpaar überdeckten Pfades in der Lattice verwendet. Die Metrik, die hierfür angegeben wird, ist lediglich die Bi-Gramm-Approximation[36] zuzüglich einer initialen Wahrscheinlichkeit für das erste Wort des Pfades[37].

Definition 3.2 *Metrik zur Berechnung von Kantenbewertungen, wie bei Chien et al. angegeben:*

$$P(W_n) = P(W_a W_i) = P(w_{a1}) * \prod_{2 \leq k \leq m} P(w_{ak}|w_{ak-1}) * P(w_{i1}|w_{am}) * \prod_{2 \leq k \leq m} P(w_{ik}|w_{ik-1}) \tag{3.2}$$

Beim Verfahren von Chien et al. sind also auf der Agenda immer alle Kanten nach Bi-Gramm-Wahrscheinlichkeiten geordnet. Im Experiment vergleichen sie nun Bestensuche nur aufgrund von Wahrscheinlichkeiten und Bestensuche mit zusätzlicher ad hoc Längenpräferenz. Selbstverständlich ist die zweite Variante wesentlich effizienter, aber fehleranfälliger. Dies erklärt sich wie folgt:

Wenn nur absolute Wahrscheinlichkeiten des Bi-Gramms verglichen werden, sind alle längeren Pfade unwahrscheinlicher als alle kürzeren. Dies ist eine Eigenschaft der Statistik und grundsätzlich korrekt. Indem die Bestensuche diese Werte als Präferenzkriterium benutzt, werden also kürzere Wortketten bevorzugt. Es resultiert ein starker Breitensucheffekt[38]. Um den Effekt zu verhindern, wird ein zusätzliches heuristisches Kriterium hinzugefügt, daß eine starke Tiefensuchtendenz einführt. Indem jedoch immer nur die längsten Pfade betrachtet werden, und davon dann diejenigen,

[35] In Hauenstein & Weber 1994 sind kleine Teile des Abschnitts 4.5 bereits veröffentlicht.

[36] Siehe hierzu die Einführung, Kap. 2.1.3.

[37] Wobei wir nur vermuten können, daß es sich hier um einen Dummy-Marker oder eine Uni-Gramm-Wahrscheinlichkeit handelt. Dies wird von Chien et al. 93 nicht weiter erläutert.

[38] Das resultierende Suchschema korrespondiert mit der *Uniform Cost Search* in der Terminologie von Shapiro 1992 [84]. Eine solche Suche garantiert zwar das Auffinden des besten Pfades, ist aber kritisch in Bezug auf Speicheraufwand. Bei größeren Unifikationsgrammatiken ist daher solches Verfahren nicht mehr praktikabel.

mit der größten Wahrscheinlichkeit, verliert das Suchverfahren die Eigenschaft, die beste Lösung zuerst zu finden.

Chien et al.1993 verzichten auf jede Art von Normalisierung der verwendeten Wahrscheinlichkeiten im Verhältnis zur Länge oder einer oberen Schranke für die Güte der Beobachtung. Daher ist ihre Vorgehensweise nicht sinnvoll. Im nächsten Abschnitt wird entwickelt, wie eine solche Metrik, die ein statistisches Language-modell und Unifikationsgrammatiken lose koppelt, aussehen kann.

Direkte Ansätze zum statistischen Parsing (enge Kopplungen) werden in Abschnitt 3 nicht weiter diskutiert. Diesem Thema ist das ganze Kapitel 5 gewidmet.

3.2.6.2 Eine flexible Metrik für LR-inkrementelles ACP

Zunächst soll geklärt werden, was eine allgemeine Metrik für die Einbeziehung von Bi-Gramm-Modellen bei der Latticesuche des ACP eigentlich leisten soll:

- Alle Kanten – also auch z.B. leere aktive Kanten – sollen korrekt und fair bewertet werden können.
- Paare von Kanten, die gleichzeitig auf der Agenda liegen können, sollen echt vergleichbar sein. Dies bedeutet, daß Normalisierungen, wie sie z.B. bei Chien et al. 1993 nicht vorkommen, vorgenommen werden müssen.
- Der maximale Kontext, der zur Einführung von Kanten führt, soll dargestellt werden.
- Die Metrik soll für LR-ACP geeignet sein, aber möglichst generell genug sein, um auch für eine breite Suche eingesetzt zu werden.
- Akustische Bewertungen von Worthypothesen des Decoders sollen korrekt in die Metrik eingehen
- Die Metrik soll erweiterbar sein. Andere Quellen statistischer Information sollen bei Bedarf einfach hinzugefügt werden können.[39]

Bei der Berechnung von Bewertungen muß zunächst geklärt werden, was *Normalisierung* bedeutet. Wie in Abschnitt 2.1.4 erläutert, sind beim Decoding Pfade entweder vergleichbar, weil sie wirklich gleich lang sind (Beamdecoder), oder weil durch eine Restkostenschätzung die gleiche Länge hergestellt wird (Stackdecoding). Reine Wahrscheinlichkeiten sind für verschieden lange Beobachtungen nicht vergleichbar, da die Anzahl der Multiplikationen verschieden ist und die Werte nicht in der selben Größenordnung liegen. Dies gilt sowohl für den Vergleich verschieden langer Wortketten mit Bi-Gramm-Bewertungen als auch für verschieden lange Worthypothesen. Bei einem Viterbi-Beamdecoder[40] korrespondiert die Länge einer Worthypothese in Frames mit der Anzahl der ausgeführten Multiplikationen für HMM-Übergänge[41]. Die minimale Anforderung für den Vergleich verschieden langer Einheiten ist die Herstellung der gleichen Größenordnungen.

[39] Wie z.B. in Kapitel 5.

[40] Wie derjenige von A. Hauenstein, Univ. HH, der für die Experimente in dieser Arbeit verwendet wurde.

[41] Die Übergänge sind die längenmodellierende Komponente.

Wir unterscheiden also vorläufig Wahrscheinlichkeit und längennormalisierte Wahrscheinlichkeit. Spätestens ab der Verwendung von Kombinationen von Wahrscheinlichkeiten, die durch verschiedene Modelle gegeben werden, sprechen wir nur noch von *Bewertung*.

Definition 3.3 *Längennormalisierte (akustische) Wahrscheinlichkeit einer Worthypothesenkette:* $logP_N(W|HMM)$

$$logP_N(W|HMM) = \frac{logP(W|HMM)}{\#(Frames)} \tag{3.3}$$

Die akustische Bewertung, die in Kantenbewertungen eingeht, entspricht der längennormalisierten akustischen Wahrscheinlichkeit. Leere Kanten, bei denen im Nenner 0 auftauchen könnte, werden weiter unten diskutiert. Beim Vergleich zweier Pfade wird nunmehr akustisch verglichen, wie gut Hypothesen über einem Frame sind. Analog gehen wir bezüglich der N-Gramm-Bewertungen vor. Dabei wird natürlich auf ein Wort anstatt auf ein Frame zurückgerechnet:

Definition 3.4 *Längennormalisierte N-Gramm-Wahrscheinlichkeit einer Worthypothesenkette:* $logP_N(W|N\text{-}Gramm)$

$$logP_N(W|N\text{-}Gramm) = \frac{logP(W|N\text{-}Gramm)}{\#(Wörter) - 1} \tag{3.4}$$

In der Praxis werden akustische Modelle und Sprachmodelle selten über der gleiche Stichprobe trainiert. Außerdem sind die Modelle bezüglich der Anzahl der trainierten Parameter verschieden. Dies führt zur Unvergleichbarkeit solcher Bewertungen. Wir führen daher ein Gewicht γ ein, um dieses Problem zu lösen. Für ein gegebenes Paar von statistischen Wissensquellen muß ein solches Gewicht eingeregelt werden[42].

Definition 3.5 *Kombinierte normalisierte Bewertung (KNB) einer Worthypothesenkette mit Gewichtung:* $logP_{NG}(W|N\text{-}Gramm,HMM)$

$$logP_{NG}(W|N\text{-}Gramm,HMM) = logP_N(W|HMM) + \gamma * logP_N(W|N\text{-}Gramm) \tag{3.5}$$

Bei KNB wie in 3.5 werden der akustische Anteil und der N-Gramm-Anteil der kombinierten Bewertung zuerst normalisiert und dann kombiniert. Dieses Vorgehen ist einerseits nicht zwingend, andererseits sogar eher ungewöhnlich. Wenn in einem Decoder N-Gramm-Übergänge direkt beim Beginn eines neuen Wortmodells aufaddiert werden, würde dies eher der folgenden Art der Kombination entsprechen:

[42] In den Experimenten dieser Arbeit ist das Gewicht zwischen Speech- und Languagemodell konstant gehalten worden. Es wurde dabei von Hand gesetzt. Durch eine Hillclimbing-Prozedur sollte für ein gegebenes System eine Optimierung erreicht werden können. Wir haben dies nicht experimentell versucht, sind jedoch der Ansicht, daß es sich um ein erforschenswertes Architekturproblem für die Zukunft handelt.

Definition 3.6 *Normalisierte kombinierte Bewertung (NKB) einer Worthypothesenkette mit Gewichtung:* $logP_{NG}(W|N\text{-}Gramm,HMM)$

$$logP_{NG}(W|N\text{-}Gramm,HMM) = \frac{\gamma * logP(W|N\text{-}Gramm) + logP(W|HMM)}{(\#(Wörter) - 1) + (\#(Frames))} \quad (3.6)$$

Wir wollen zwei Wortketten W und V betrachten, die über k Frames hypothetisiert wurden. W besteht aus vielen kurzen Wörtern, während V nur aus wenigen langen Wörtern besteht.

Falls bei der Viterbi-Berechnung der akustischen Bewertung immer beim Wortübergang die N-Gramm Bewertung aufaddiert wird, so entsteht eine kombinierte Bewertung für W und V aus verschieden vielen Additionen, da durch die Wortübergangsstrafe kein Frame konsumiert wird.[43] Der durch das N-Gramm gegebene Anteil an der Bewertung von V ist wesentlich niedriger als bei W.

Es gibt mehrere Argumente, KNB zu verwenden anstatt NKB:

- Bei der Kopplung des Parsers mit einem Beamdecoder im nächsten Abschnitt wird die N-Gramm-Wahrscheinlichkeit – zumindest bei der bottom up Kopplung – nur vom Parser verwendet. Für Wortübergänge im Decoder wird immer eine konstante Wortübergangsstrafe verwendet, die jedoch nicht mit an den Parser übertragen wird. Für zwei Ketten W und V, die vom Decoder hypothetisiert wurden, wurden also immer gleich viele Additionen ausgeführt, bevor die N-Gramm Bewertung im Parser hinzugefügt wird.

- Wenn, wie in Kapitel 5, noch Ableitungswahrscheinlichkeiten der Unifikationsgrammatik mit in die kombinierte Bewertung aufgenommen werden, sollen größere Ableitungsbäume nicht zu einem größeren Gewicht des Grammatikscores führen. Spätestens hier deckt sich KNB mehr mit der Intuition als NKB.

- Die Verwaltung von KNB ist wesentlich einfacher. Beim inkrementellen Time Map[44] werden teilweise akustische Anteile von Bewertungen einer Update-Operation unterworfen. Indem alle Anteile getrennt gehalten werden, und nur vor Vergleichen zusammenaddiert werden, sind solche Operationen wesentlich billiger zu implementieren.

- KNB mit einem bestimmten Gewicht γ entspricht der Annahme, daß der N-Gramm-Anteil für ganze Äußerungen immer konstant ist, in anderen Worten: Die Anzahl gesprochener Wörter für einen Abtastzeitraum variiert nicht sehr stark im Bereich ganzer zu parsender Äußerungen.[45]

- Im Experiment nur mit akustischem und N-Gramm Bewertung konnte von uns kein Unterschied in der Erkennungsleistung festgestellt werden.

Bisher sind zwei der geforderten Punkte (beinahe) erfüllt: N-Gramm und akustischee Bewertung können integriert werden. Darüber hinaus könnten wir zumindest passive Kanten über beliebigen Pfaden vergleichen. Um alle Forderungen zu

[43] Vergl. Ney 1993 [66].

[44] Siehe Kapitel 4.6.4.2.

[45] Möglicherweise trifft das nur für die von uns verwendeten Sprachdaten zu. Bekanntlich variiert die Sprechgeschwindigkeit bei spontaner Sprache wesentlich stärker als bei gelesener.

erfüllen, teilen wir die Bewertung von Kanten auf in eine *INSIDE-Bewertung* und eine *OUTSIDE-Bewertung*.[46]

Definition 3.7 *INSIDE-Bewertung einer Kante K, die W überspannt: $PI(K)$*

$$PI(K) = \begin{cases} logP_{NG}(W|N\text{-}Gramm,HMM) & ,falls\ length(W) > 0 \\ 0 & ,falls\ length(W) = 0 \end{cases} \tag{3.7}$$

Die INSIDE-Bewertung geht also für alle Kanten aus dem Pfad in der Lattice hervor, den sie unmittelbar überspannen. Nur die leeren aktiven Kanten erhalten per Definition den Wert 0. Somit kann 0 nicht im Nenner von Gleichung 3.3 oder 3.4 erscheinen.

Die OUTSIDE-Bewertung richtet sich nach der Art der Kanteneinführung des Parsingalgorithmus. Sie soll die Evidenz aus dem Kontext widerspiegeln, mit der es dazu kam, daß eine Kante entstehen konnte. Während bei einem bottom up Parsingschema neue aktive Kanten nur aufgrund von terminalen Ereignissen eingeführt werden, ohne daß der sonstige Zustand der Chart eine Rolle spielt, ist der Kontext außer der jeweiligen Worthypothese leer. Die Bewertung der Worthypothese jedoch spiegelt sich bereits voll in der dazugehörigen INSIDE-Bewertung der passiven Kante wieder. Im Falle von BU-Parsing ist also die OUTSIDE-Bewertung einer leeren Kante 0. Auch bei allen anderen Kanten gilt im Falle der BU-Verarbeitung, daß der Kontext aufgrund dessen eine Kante eingeführt wird, immer nur der überspannte Pfad ist.

Definition 3.8 *OUTSIDE-Bewertung einer Kante K, die W überspannt für BOTTOM-UP Parsingschema: $PO(K)$*

$$PO(K) = PI(K) \tag{3.8}$$

Im Falle von top down Parsing wird die Einführung leerer Kanten ausgelöst durch bereits bestehende aktive Kanten während der SEEK-DOWN Operation. Die einführende aktive Kante konnte wiederum selbst nur durch einen entsprechenden linken Kontext in der Chart entstehen. Wir definieren also die OUTSIDE-Bewertung bei top down Verarbeitung rekursiv, ausgehend von der Startkante in Knoten 0, die mit dem Wert 0[47] initialisiert wird, für leere aktive Kanten.

Definition 3.9 *OUTSIDE-Bewertung einer Startkante K_{init}, für TOP DOWN Parsingschema: $PO(K_{init})$*

$$PO(K_{init}) = 0 \tag{3.9}$$

[46] Die Namen "INSIDE-" und "OUTSIDE-" übernehmen wir vom Inside-Outside-Algorithmus (Baker 1979 [8] oder Jelinek 1992 [44]). Der IOA verwendet für die Kalkulation von Ableitungswahrscheinlichkeiten einen *Outside-Score* jedoch für den gesamten Kontext rechts und links von einer Variablen, die einen Teilbaum dominiert. In unserer Metrik wird von der OUTSIDE-Bewertung nur der linke Kontext dargestellt, wie ein einem LR-inkrementellen Verfahren nicht anders zu erwarten. Päseler's 1988 [67] IS und AS korrespondieren eher mit den hier verwendeten Bewertungen, enthalten jedoch nur akustische Modellwahrscheinlichkeiten.

[47] Wahrscheinlichkeit 1.

Definition 3.10 *OUTSIDE-Bewertung einer leeren inaktiven Kante K_j, für TOP DOWN Parsingschema: $PO(K_j)$*

$$PO(K_j) = PO(K_i),\ gdw. K_j\, eingeführt\ wird\ von K_i \tag{3.10}$$

Innerhalb eines Knotens erhalten also alle leeren aktiven Kanten die OUTSIDE-Bewertung derjenigen Aktiven, die zu ihrer Einführung führte. Wenn die OUTSIDE-Bewertung den linken Kontext einer Kante wiederspiegeln soll, dann heißt das im vorliegenden Fall, daß der bis zu dieser Kante bereits geparste Pfad, der dahin führte, in die Bewertung eingehen muß. Wir erreichen dies, indem wir beim Kombinieren von aktiven und passiven Kanten die OUTSIDE-Bewertung der neuen Kante aus der OUTSIDE-Bewertung der aktiven und der INSIDE-Bewertung der passiven zusammensetzen. Dadurch wird immer die ganze Bewertung von Knoten 0 an bis zum aktuellen Verarbeitungszeitpunkt aufgesammelt, unabhängig von aktuellen Span der beiden involvierten Kanten. Bei einer Anwendung der Fundamentalen Regel wird also die INSIDE-Bewertung rekursiv berechnet wie durch die Definitionen 3.3, 3.4, 3.6 und 3.7 gegeben.[48] Um das N-Gramm korrekt in die Berechnung zu integrieren, führen wir für jede Kante den Pfad von Worthypothesen durch die Lattices mit, aufgrund derer sie eingeführt wurde:

Definition 3.11 *Words(K) ist die Kette von Worthypothesen, die von K direkt überspannt wird.*

Definition 3.12 *Rekursiver Aufbau des Pfades einer Kante: Path(K)*

1. *Der Pfad der Startkante ist der Marker \$begin\$*
2. *Bei jedem SEEK-DOWN wird der Pfad von der einführenden an die eingeführte aktive Kante vererbt.*
3. *Für jede Kombination von aktiver Kante K_i und passiver Kante K_j ist der Pfad die Konkatenation von Path(K_I) und Words(K_j).*

Wir verwenden die obige Definition des Pfades einer Kante in der Kalkulation der neuen OUTSIDE-Bewertungen. Dabei muß der Pfad zugreifbar sein, da der N-Gramm-Wert an der Stelle der Verbindung der Kanten mit in das Resultat eingehen soll.

Definition 3.13 *Berechnung der neuen OUTSIDE-Bewertung bei Anwendung der Fundamentalen Regel:*

$$\begin{aligned} PO(W_k) \quad = \quad & Normalize(\\ & Denormalize(PO(W_i)) + \\ & Denormalize(PI(W_j)) + \\ & logP(last(Words(W_j)))|(first(Path(W_i))), N - Gramm) * \gamma) \end{aligned} \tag{3.11}$$

[48] Natürlich wird in der Implementation nicht wiederholt die Rekursion ausgeführt. Der bereits berechnte Teil sowohl von INSIDE- als auch OUTSIDE-Bewertung wird aus den bereits bestehenden Kanten genommen.

Obwohl Definition 3.13 im Vergleich zu den simplen Gleichungen oben unelegant anmutet, ist sie nichtsdestotrotz korrekt. Da alle Werte von PO und PI normalisiert sind, können sie nicht direkt verrechnet werden. Die logarithmischen Darstellungen der Modellwahrscheinlichkeiten können addiert werden. Anschließend wird das Ergebnis wieder normalisiert.[49]

Denormalize – um präzise zu sein – dekomponiert INSIDE- und OUTSIDE-Bewertungen wieder in ihre akustischen und N-Gramm-Anteile und rechnet die Normalisierung zurück, sodaß die Übergangsstrafe zum N-Gramm-Anteil hinzuaddiert werden kann. *Normalize* tut genau das inverse hierzu. Beide Anteile werden wieder normalisiert und zusammenaddiert.

In der Implementierung wird für eine Kante immer die logarithmische Darstellung der Wahrscheinlichkeiten getrennt für Akustik und Sprachmodell mitgeführt. Die Anzahl überdeckter Wörter und Frames ist ebenfalls für jede Kante bekannt. Die aktuellen normalisierten Werte werden ebenfalls mitgeführt, um sie nicht für jeden Vergleich neu berechnen zu müssen. Neue Werte für neue Kanten werden schließlich in logarithmischer Darstellung von Modellwahrscheinlichkeiten berechnet, wobei Teilkalkulationen für die Kanten, die von der Fundamentalen Regel kombiniert werden, verwendet werden. Erst für die Einordnung eines Paares von Kanten auf der Agenda wird dann wieder normalisiert.

3.2.6.3 Pruning und Suche

In Abschnitt 3.2.1 ist bereits angedeutet, daß in jedem Parsezyklus der Pruning-Schritt zuerst erfolgt und die verbleibenden Parseschritte nachher ausgeführt werden. Die Details dieser Vorgehensweise und die Gründe hierfür sollen an dieser Stelle erläutert werden.

Zunächst stellt sich die Frage, wieso Pruning überhaupt nötig ist. Pruning ist keinesfalls eine Notwendigkeit im Latticeparsing. Gerade im Bereich Chartparsing ist eine Bestensuche ohne Pruning die übliche Vorgehensweise, wie z.B. bei Chien et al. 1990 und 1993 [16, 17]. Dabei würde bei einer Agenda-getriebenen Architektur, wie in 2.3.2, nur das jeweils beste Paar von Kanten auf der Agenda weiterverfolgt und nach jedem Parseschritt alle neuen sich ergebenden Paare in die sortierte Agenda eingefügt. Danach wird wieder nur das beste Paar geparst, usw.

Falls die N besten Wortkette–Analyse–Paare gefunden werden sollen, ist dies eine geradlinige Vorgehensweise. Bekanntlich ist sie jedoch an einige Voraussetzungen geknüpft. Da bei dieser Art der Suche bewertungsgetrieben in die Tiefe gesucht werden soll, sind einige Bedingungen erforderlich, die im folgenden diskutiert werden.

- Verschieden lange Pfade müssen echt vergleichbar sein. Dies kann z.B. durch die Normalisierung teilweise erreicht werden. Dennoch ist zu bedenken, daß es sich hierbei nur um eine Approximation einer echten Vergleichbarkeit handelt. Durch die Normalisierung der Anzahl von Additionen wird im Grunde nur eine gleiche Größenordnung von numerischen Bewertungen hergestellt. Über verschiedenen Signalabschnitten können dennoch, abhängig von lokalen Maxima, zwei Hypothesen mit der selben normalisierten Bewertung einmal gut und ein anderes mal schlecht sein — immer relativ zu den direkt über dem

[49] In 3.13 wird dafür die intuitiv zugängliche Funktionenschreibweise benutzt.

selben Abschnitt konkurrierenden Hypothesen. Bereits bei Woods 1982 [97] wird durch die Shortfall-Methode ein Verfahren vorgestellt, in einem Chartparser alle Bewertungen in Verhältnis zu lokalen Maxima zu relativieren. Die Voraussetzung hierfür ist jedoch, daß die besten Hypothesen immer zuerst eintreffen. Dies ist nicht mit einer LR-inkrementellen Architektur vereinbar.

- Um z.B. eine A*-Suche durchführen zu können, muß dem Parser der gesamte Wortgraph vorliegen und eine optimistische Restkostenschätzung vorgenommen werden können. Bei einer LR-inkrementellen Vorgehensweise ist die erste Bedingung theoretisch nie erfüllt. Die zweite Forderung kann natürlich für die N-Gram-getriebene Suche erfüllt werden, indem der Anteil der Restkostenschätzung auf 0 gesetzt wird. Im Restultat entspricht dies aber einer *Uniform–Cost–Suche*[50], obwohl die Normalisierung dafür sorgt, daß kein zusätzliches Tiefenkriterium nötig ist, wie bei Chien et al. 1993[51]

- Eine Restkostenfunktion, die wirklich optimistisch schätzt — also eine möglichst gute untere Schranke der Kosten angibt, kann in einem unifikationsbasierten Parser nicht zur Verfügung stehen. Während der Suche werden Pfadverlängerungen immer auch durch grammatische Operationen verifiziert. Das heißt, daß der noch unbekannte Teil eines Pfades immer grammatisch scheitern kann — also die Kosten unendlich groß werden[52]. Die Güte der Restkostenfunktion kann also nicht garantiert werden. Praktisch gesehen heißt dies, daß zahlreiche Pfade bis kurz vor dem Äußerungsende "top" sein können, jedoch im nächsten Schritt vollständig an einer scheiternden Unifikation beendet werden müssen.

Verschiedene Experimente, in denen der Parser über bereits eingelesenen Wortgraphen eine Bestensuche ausführt, wobei alle Kantenpaare über beliebigen Signalabschnitten miteinander verglichen werden, führten zu guten Ergebnissen. Durch die Normalisierung der Bewertungen wird ein Effekt erreicht, als wäre die Restkostenschätzung bei A* immer entsprechend der durchschnittlichen Bewertung des bekannten Abschnitts. Dies führt theoretisch dazu, daß die beste Lösung nicht zuerst gefunden werden muß. Der Fall trat wirklich in einem von zehn Fällen auf, wobei die korrekte Lösung jedoch nie unterhalb von Rang 3 lag.[53]

Indem wir jedoch durch die Kontrollschleife in Abschnitt 3.2.1 ein striktes LR-Kriterium einführen, wird diese Suche, wie ja bereits oben angeführt, obsolet.

Wenn bei einer LR-inkrementellen Suche und Analyse alle Pfade zeitsynchron verfolgt werden, resultiert – auch wenn Bewertungen verwendet werden, ein Breitensucheffekt. Dies ist jedoch aufgrund der Größe des Suchproblems prohibitiv.

In links-rechts orientierten Verfahren in der Spracherkennung sind Beam-Search-Verfahren bereits erfolgreich eingesetzt worden, wie z.B. im Sphinx–System von Lee 1989 [57], oder bei Ney 1993 [66]. Dies ist bei zeitsynchronen Pfadsuchproblemen eine angemessene Vorgehensweise, wenn keine gute Restkostenschätzung verfügbar ist.

[50] Vergleiche Shapiro 1992 [84] und dort zitiert Pohl 1970 [71] und Davis et al. 1989 [20].

[51] Der Unterschied liegt hier nicht mehr in der Suchstrategie. Durch die Normalisierung wird lediglich die Scoring–Funktion gerechter als in [17].

[52] Was heißt, daß der Zielzustand nie erreicht werden kann.

[53] Hierbei ging es nicht um eine *Erkennungsrate*, sondern nur um die Vollständigkeit des Suchverfahrens.

Päseler 1988 [67] hat bereits eine solche Vorgehensweise für den Earley-Algorithmus, angewendet auf Mengen von Worthypothesen, vorgeschlagen. Nur unter Verwendung von akustischen Bewertungen von Worthypothesen und einer kontextfreien Grammatik werden dort gute Ergebnisse erzielt. Hier wird jedoch direkt auf der Liste von Worthypothesen für einen Parsezyklus ein Pruning vollzogen. Da bei Päseler 1988 nur mit akustischen Bewertungen gearbeitet wird, ist das Pruning aufgrund dieser lokalen Information gerechtfertigt. Im vorliegenden Ansatz wird jedoch zusätzlich mit statistischer Language–Information gearbeitet. Es kann also für ein Wort entschieden werden, wie gut es in einem bestimmten linken Kontext wird. Beim Pruning soll daher die Möglichkeit bestehen, daß Worthypothesen im Kontext von bestimmten Suchpfaden unter den Beam fallen, jedoch in anderen Kontexten nicht. Wir realisieren das Pruning daher als Nichtbeachten von Kantenpaaren der Agenda, und nicht vor dem Scanning[54], als Verwerfen von Worthypothesen. Folglich können wir Worthypothesen in einem gutbewerteten Kontext weiter verfolgen, dieselbe Hypothese in einem anderen Kontext jedoch verwerfen.

Bei LR-ACP wird nach dem Einlesen der Worthypothesen für jede Hypothese pro Lesart im Lexikon eine passive Kante in die Chart eingefügt. Für jede dieser Kanten wird ein Paar mit jeder adjazenten aktiven Vorgängerkante auf die Agenda gestoßen. Nach dem Einlesen enthält die Agenda eine Liste aller unmittelbar möglichen ersten Operationen von disjunkten Folgen von Completer-Operationen. Da alle aktiven Kanten von Paaren aufgrund der Bestensuche beim Parsing die Bewertungen der besten Pfade tragen und damit eine untere Kostenschranke darstellen, können die nachfolgenden jeweiligen Completerschritte nicht besser werden. Sie werden also durch ihre prototypische Anfangskonfiguration auf der Agenda voll repräsentiert. Es kann nicht passieren, daß ein Kantenpaar, daß beim Pruning verworfen wurde, später im Completerschritt besser würde.

Flexibles Pruning erreicht man durch einen Offset vom lokalen Maximum, wie in Päseler 1988 [67] oder Lee 1989 [57]. Wir addieren also in jedem Zyklus einen Offset auf das bestbewertete Paar der Agenda. Die Agenda wird schließlich geteilt in die Hälfte oberhalb der Beamschranke und diejenige unterhalb. Die folgenden Voraussetzungen sind gegeben:

- BEAMVALUE ist der Parameter für die Weite des Suchstrahls
- *Score* gibt für ein Paar von aktiver und passiver Kante auf der Agenda dessen Bewertung zurück. Die Bewertung eines Paares ist immer die Bewertung der neuen Kante, die bei einer erfolgreichen Kombination entstünde, wie in Abschnitt 3.2.6.2.
- Die Agenda wird immer sortiert gehalten, d.h. ein neues Paar wird immer gleich einsortiert.[55]

Das eigentliche Parsingverfahren für einen Zyklus kann dann wie folgt beschrieben werden:

[54] Welches im Übrigen als eigenständige Operation beim unifikationsbasierten ACP gar nicht existiert, da ja bei stark lexikalisierten Unifikationsgrammatiken die Terminale der Grammatik Merkmalsstrukturen sind und nicht Wörter. Ein explizites Lexikon für die kontextfreie Grammatik hingegen existiert nicht bei Päseler 1988.

[55] Bekanntlich erfolgt dies binär in $O(\log n)$

Definition 3.14 *LR-ACP mit Strahlensuche nach rechts.*

1. *Für alle ankommenden Worthypothesen in einem Zyklus: Füge W_i in die Chart ein, sodaß alle Paare von aktiven Kanten mit W_i auf die Agenda gestoßen werden.*

2. *BESTVALUE sei MAX(Score(active, W_i)), die Bewertung des Top-Paares auf der AGENDA.*

3. *PRUNEVALUE := BESTVALUE + BEAMVALUE*

4. *Teile die Agenda in PARSE-AGENDA und PRUNE-AGENDA, sodaß der Wert des letzten Paares von PARSE-AGENDA noch besser ist als PRUNEVALUE.*

5. *Parse PARSE-AGENDA*

6. *Save-Agenda*

Wie bei Päseler 1988 wird die Strahlensuche hier nur nach rechts ausgeführt. Ein Unterschied zwischen dem Ansatz in Algorithmus 3.14 und Päselers Ansatz besteht nur in der Miteinbeziehung eines größeren Kontextes um die Bewertung für das Pruning vorzunehmen.

In Algorithmus 3.14 wird zuerst die gesamte aktuelle Agenda für einen Parsezyklus gebildet. Für alle Paare, die auf PARSE-AGENDA entfallen, also oberhalb des Beams liegen, wird die vollständige Completeroperation ausgeführt. Genau hier liegt jedoch die Schwäche gerade des Päseler-Ansatzes:

Während die Completeroperation — also die Rückwärtssuche im Parser — alleine bereits quadratisch ist und das Parsing also kubisch, ist die Rückwärtssuche im Decoder viel billiger.[56] Ein solches Vorgehen im Hinblick auf die enge Kopplung von Parser und Decoder, bringt also bezüglich der Komplexität starke Nachteile. Eine traditionelle bottom up Architektur wäre hier immer überlegen.

Gerade durch das Pruning über der Agenda ist jedoch beim LR-ACP im Unterschied zu Päseler die Möglichkeit gegeben, auch beim Completerschritt eine Strahlensuche auszuführen.

Um die Strahlensuche im LR-ACP bei der Suche nach rechts und der Suche nach links einzusetzen, darf das Pruning nicht mehr nur strikt über der initialen Agenda eines Zyklus stattfinden. Die Agenda bleibt bis zum Ende des Zyklus ungeteilt, wird jedoch nur bis zur Beamschwelle zugegriffen. Wenn also während der Completeroperation neue Paare auf die Agenda gestoßen werden, können diese ebenfalls noch unter die Suchschranke fallen.

Die Save-Agenda-Operation, die unten noch weiter erläutert wird, rettet schließlich die Agenda, wenn alle aktuellen Paare während der Completeroperation unter die Schwelle gefallen sind.

Definition 3.15 *LR-ACP mit Strahlensuche nach rechts und links.*

[56] Da bei Päseler noch eine kontextfreie Grammatik verwendet wurde, fiel der Completerschritt nicht so teuer aus. Durch die Verwendung einer Unifikationsgrammatik wird dieses Problem erst prominent.

1. *Für alle ankommenden Worthypothesen in einem Zyklus: Füge W_i in die Chart ein, sodaß alle Paare von aktiven Kanten mit W_i auf die Agenda gestoßen werden.*

2. *BESTVALUE sei $MAX(Score(active, W_i))$, die Bewertung des Top-Paares auf der AGENDA.*

3. *PRUNEVALUE := BESTVALUE + BEAMVALUE*

4. *Parse AGENDA, bis der Wert des Top-Paares kleiner als PRUNEVALUE ist.*

5. *Save-Agenda*

In den Experimenten, die in Abschnitt 4.5 referiert sind, wurde für den experimentierten Bereich ein lineares Verhältnis von Worthypothesen zum Arbeitsaufwand erreicht.

Die Operation *Save-Agenda* besteht darin, in jedem Zyklus die verbleibende Agenda unterhalb der Suchschwelle in einer globalen Liste aufzuheben.

Indem wir alle Paare aufheben, die bei der Strahlensuche unterhalb des Suchstrahls liegen, kann im Falle des Fehlschlagens der Strahlensuche der verbleibende Suchraum noch weiter exploriert werden. Die Gründe sind naheliegend:

Zur Bewertung von Suchpfaden vor dem Pruning wird nur die statistische Komponente verwendet[57]. Es kann also der Fall auftreten, daß alle Suchpfade, die einen Beamschritt überlebt haben, später durch ein Scheitern von Unifikationsoperationen absterben. In diesem Falle wird kein Ergebnis gefunden. Um zu garantieren, daß das beste parsbare Ergebnis gefunden wird, sofern die gesamte Menge von Worthypothesen ein solches enthält, sehen wir vor, bei fehlschlagendem Beamsearch noch über der ganzen Chart weitersuchen zu können.

- Parse zeitsynchron bis das erste (beste) Resultat gefunden wird.

- Falls kein Resultat gefunden wird, parse die globale PRUNE-AGENDA, bis ein Resultat gefunden wird.

Damit ist die Möglichkeit gegeben, effizient zeitsynchron zu parsen bzw. zu suchen, jedoch im Falle eines Fehlschlages ein *Recovery* vorzunehmen. Das Verfahren wird so robust gegenüber den Fehlern, die durch die Verknüpfung von statistischer und symbolischer Restriktion auftreten können.

Es bleibt anzumerken, daß die Flexibilität, mit der hier vorgegangen werden kann, durch die Abstraktion von Daten und Kontrolle ermöglicht wird, die dem aktiven Chartparsing inhärent ist.

[57] Dies ändert sich auch nicht, falls – wie in Kapitel 5 – die Unifikationsgrammatik trainiert wird.

Kapitel 4

Interaktion mit einem Beam-Decoder

Bisher ist ein Parsingmodell vorgestellt worden, daß im wesentlichen die folgenden Eigenschaften vereint:

- Parsing erfolgt über Lattices. Grundsätzlich kann jede auf einem Zeitraster verankerte Menge von Worthypothesen verarbeitet werden. Eine erfolgreiche Suche kann erfolgen, wenn die Eingabe ein von links verbundener Wortgraph ist. Letzteres wird in Abschnitt 4.6.1 noch detaillierter untersucht.
- Die Analyse erfolgt mit einer Unifikationsgrammatik. Dies ermöglicht die Kopplung mit einer "echten" Semantik. Wie z.B. in HPSG [72] kann hier Syntax und Semantik in den selben Regeln verarbeitet werden.
- Die Suche und Analyse erfolgt simultan.
- Suche und Analyse sind lose gekoppelt. Der Suchraum wird bewertet durch akustische Scores, N-Gram Scores und Parsbarkeit der Unifikationsgrammatik. Bei Ausführen der Suche werden alle drei Restriktionsarten gleichzeitig angewendet.
- Die Suche nach rechts ist unvollständig. Nur aufgrund der probabilistischen Modelle plausible Suchpfade werden weiterverfolgt.
- Die Suche nach links ist unvollständig. Die Strahlensuche nach rechts und nach links ist einheitlich über die Agenda gesteuert.
- In einer optionalen zweiten Suchphase ist die Suche vollständig. Das Verfahren wird hierdurch robust.[1]

Wichtig für ein Verarbeitungsmodell ist jedoch auch die Einbettung in einen Systemzusammenhang. Nur durch den Einsatz in einem "echten" System sind die

[1] Insbesondere robust gegen einen zu engen Beam.

Vor- und Nachteile eines Verarbeitungsmodells korrekt zu beurteilen. In diesem Abschnitt werden daher verschiedene Kopplungen mit einem Viterbi Beam Decoder[2] beschrieben. Der Schwerpunkt liegt dabei auf zwei verschiedenen Arten der Kopplung mit top down Datenfluß, die beide LR-inkrementell sind. Hierfür sind spezielle Erweiterungen des LR-ACP nötig, die vorgestellt und diskutiert werden.

4.1 LR-inkrementelle Architektur

In der Literatur sind einige psycholinguistisch motivierte Modelle des integrierten Sprachverstehens vorgeschlagen worden, wie z.B. Briscoe 1987 [12] oder Görz & Weber 91 [38]. Diese Modelle haben immer gemeinsam, daß verschiedene Verarbeitungskomponenten, die für das Modell angenommen werden, parallel und zeitsynchron im Sinne der Sprechrichtung arbeiten.

Obwohl solche und vergleichbare Arbeiten immer richtungsweisend für die vorliegende Arbeit waren, haben sie ohne Ausnahme die folgenden Mängel:

- Anwendungsbeispiele sind immer konstruiert.
- Die verwendeten Techniken sind exotisch und empirisch nicht getestet.
- Tests auf echten Daten sind meistens nicht dokumentiert. Es ist zu vermuten, daß sie nie durchgeführt wurden.

Mit dem LR-ACP ist die Möglichkeit vorhanden, eine zeitsynchrone interaktive Kopplung mit einem Decoder zu realisieren und zu testen. Der Grund hierfür ist, daß die "Ingredienzien" des LR-ACP Standardtechnologien darstellen.

In einer bottom up Standardarchitektur werden von der Akustik bis zur Semantikkonstruktion drei Wissensquellen angewendet:

Hidden Markov Modelle zur Suche der besten Wortkandidaten

N-Gramm-Modelle zur Suche der besten Wortkette

Unifikationsgrammatiken zum Aufbau einer syntaktisch-semantischen Beschreibung

Bei den verschiedenen Kopplungen, die in diesem Kapitel beschrieben sind, werden ebenfalls nur diese Wissensquellen angewendet. Die Reihenfolge, in der Beschränkungen durch die Wissensquellen angewendet werden, ist jedoch verschieden.

Neben den psycholinguistisch motivierten Modellen gibt es bereits eine kleine Zahl von Arbeiten, die technisch ausgearbeitet sind. Hier sind vor allem Murveit et al 90 [62] und Dupont 93 [23] zu nennen, aber auch bereits Päseler 1988 [67]. Neuere Untersuchungen mit stochastischen CFGs finden sich auch in Jurafsky et al. 1994 [48]. In all diesen Arbeiten werden jedoch als Formalismus zur symbolischen Sprachbeschreibung nur endliche Automaten bzw. kontextfreie Grammatiken eingesetzt[3] . Realistische Sprachbeschreibungen, auch über eingeschränkten Domänen, können

[2] Der verwendetet Decoder stammt von Andreas Hauenstein, Universität Hamburg. Siehe auch Hauenstein & Weber 1994a und 1994b [39, 40].

[3] In Jurafsky et al. 1994 wird in einer Fußnote angemerkt, daß sie ebenfalls beabsichtigen, in zukünftiger Forschungsarbeit zu merkmalsbasierten Sprachbeschreibungen überzugehen.

jedoch auf der Basis solcher Formalismen nicht erreicht werden. Insofern sind die dort beschriebenen Ergebnisse zwar ermutigend, aber noch nicht wirklich aufschlußreich. Wir präsentieren in diesem Abschnitt verschiedene LR-inkrementelle Kopplungen von Decoder und Parser, wobei nur *Standardtechnologie* verwendet wird, wie sie auch in ausgearbeiteten seriellen Systemen verwendet wird. Der Schritt zur interaktiven Kopplung mit einer Unifikationsgrammatik ist auch der Schritt in Richtung einer realen Anwendung für maschinelle Übersetzung oder Auskunftssysteme, durch den Anschluss an eine echte Semantik, die über Spielanwendungen hinausgehen kann.

4.2 Bottom Up Inkrementelle Kopplung

Die einfachste Art der Kopplung, die sich LR-inkrementell verhält, ist die bottom up inkrementelle Kopplung (BUI). Diese Art der Kopplung ist nicht interaktiv, d.h. der Datenfluß geht nur in eine Richtung — vom Decoder zum Parser.

Das Verhalten des Beamdecoders ist dabei insofern abweichend von einer Standardimplementierung, als jede gefundene Wortendehypothese sofort als Worthypothese an den Parser geschickt wird. Da der Beamdecoder frameweise von links nach rechts entlang der Zeitachse der Eingabe arbeitet, kommen alle Worthypothesen mit dem gleichen Endezeitpunkt auf einmal. Im Experiment schickt der Decoder nach jedem Zyklus ein Synchronisationssignal an den Parser.

Die a priori gegebenen Eigenschaften dieser Art der Kopplung sind folgende:

- Der Decoder kann, nachdem er das Synchronisationssignal gesendet hat, weiter arbeiten, während der Parser ebenfalls arbeitet. Logisch gesehen ist also eine echte Parallelität realisiert.
- Bis auf die Eigenschaft, Worthypothesen sofort auszugeben, bleibt der Decodingalgorithmus unverändert.
- Der Decoder verwendet nur das akustische Modell.
- Der LR-ACP wird direkt — wie bisher beschrieben — eingesetzt.
- Obwohl beide Komponenten synchron arbeiten, sind sie bezüglich der Information, die sie verarbeiten, abgekapselt.

Die Wissensquellen sind bei BUI wie in Abbildung 4.1 auf die Komponenten verteilt. Während die Suche im Decoder nur durch das akustische Modell gesteuert wird, ist beim LR-ACP das Suchverhalten durch das akustische Modell, das statistische Languagemodell und die Unifikationsgrammatik gesteuert. Der Datenfluss, in Abbildung 4.1 durch die Pfeile dargestellt, geht nur in Richtung des Parsers.

Die bereits erwähnte Parallelisierbarkeit von BUI ist deren großer Vorteil gegenüber nicht inkrementellen Verfahren einerseits, aber auch den top down Varianten, die im folgenden vorgestellt werden. Indem die BUI Kopplung keine Antworten des einen Moduls an ein anderes vorsieht, um etwa dessen Suchverhalten zu beeinflussen, muß auch nicht gewartet werden. Hinzu kommt, daß der Parser üblicherweise das langsamere Modul ist[4]. In den Experimenten in Abschnitt 4.5 sind daher beide Module als je ein Prozeß auf einem eigenen Prozessor realisiert.

[4] Standardalgorithmen für Decoding sind $O(n^2)$, die für Parsing sind $O(n^3)$.

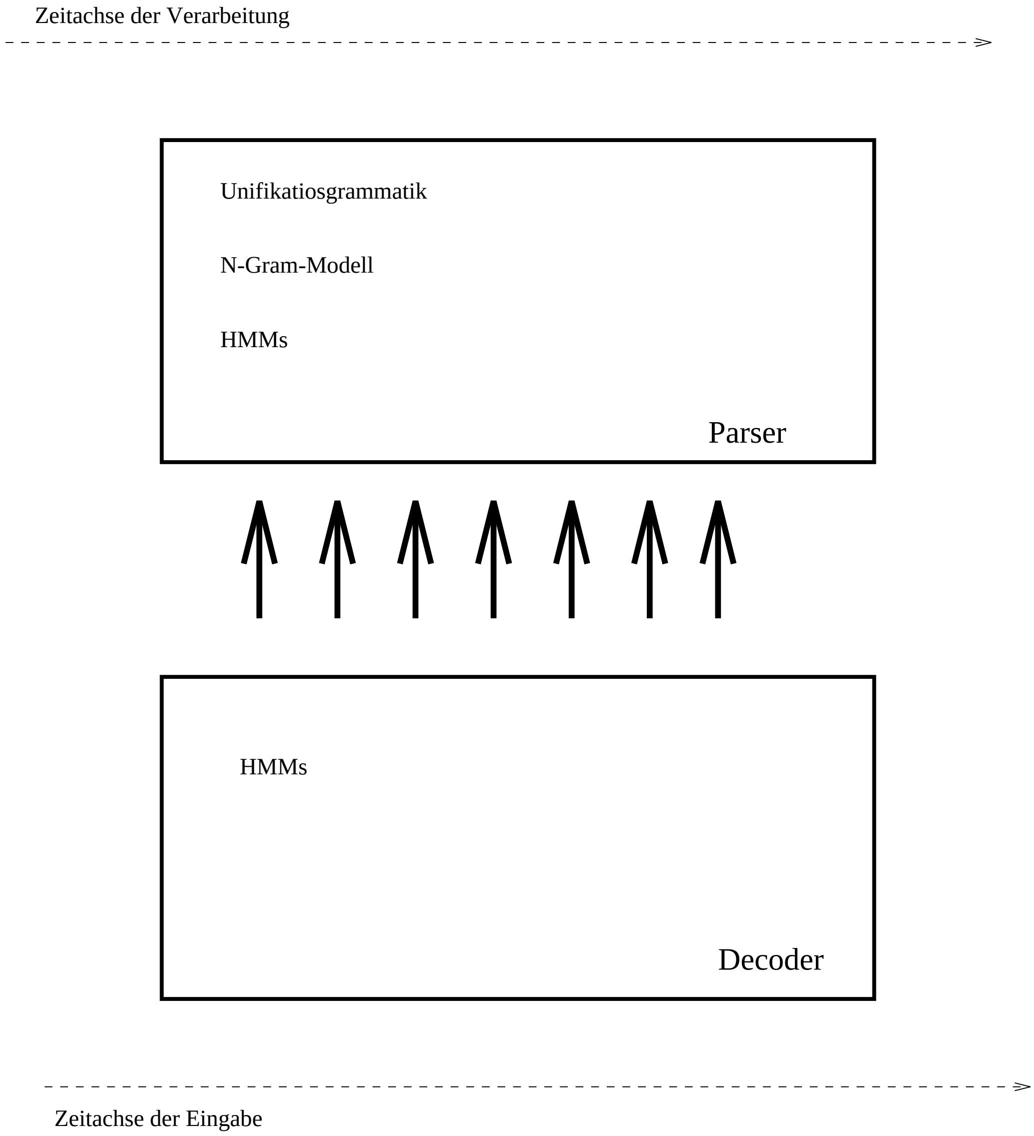

Abbildung 4.1: Parser und Decoder bei BUI

Durch die Kopplung bei BUI erhalten wir ein System mit zwei unabhängigen Suchstrahlen. Dies ist grundsätzlich mit der bottom up Vorgehensweise verträglich.

4.3 Bottom Up Inkrementelle Kopplung mit Top Down Verifikation

In einer zeitsynchronen interaktiven Architektur gibt es zwei Möglichkeiten, Parser und Decoder zu koppeln. Die Kontrolle liegt jeweils bei einem der Module. Die bottom up inkrementelle Kopplung mit top down Verifikation (BUITDV) ist der Fall, bei dem der Decoder die Kontrolle hat.[5]

Wenn ein Beamdecoder nur die akustischen Modelle verwendet, dann wird beim Wortübergang eine Strafe aufaddiert, die eine Inflation von Wortanfängen vermeidet. Falls im Decodingschritt bereits das statistische Sprachmodell mitberücksichtigt wird, dann wird genau diese Strafe durch die vom Sprachmodell gegebene Wortübergangsstrafe ersetzt.

Da in einem LR-inkrementellen System der LR-ACP schritthält, steht jedoch noch mehr Information zum Zeitpunkt eines möglichen Wortübergangs zur Verfügung.

Der LR-ACP kann für jede gesendete Worthypothese entscheiden,

- ob es einen Pfad von Worthypothesen gibt, für den die aktuelle Hypothese eine parsbare Verlängerung darstellt.
- und wie die Language-Strafe für den besten parsbaren Vorgänger ist.

Indem diese Information als top down Nachricht an den Decoder zurückgesendet wird, erhält der Decoder nicht nur statistische Languageinformation sondern auch symbolische Languageinformation aus der Beschränkung der Unifikationsgrammatik.

Auf der Seite des Decoders soll die vom Parser zur Verfügung gestellte Languageinformation als Wortübergangsstrafe in die LR Viterbisuche integriert werden. Für jede begonnene Wortkopie des Decoders muß dies jedoch nur einmal stattfinden. Das heißt, das jede bottom up an den Parser gesendete Hypothese entweder noch zu verifizieren ist, oder die entspechende Parserinformation dem Decoder bereits bekannt ist. Jede bottom up gesendete Worthypothese wird hierfür mit einer entsprechenden Markierung ausgestattet. Auf der Seite des Parsers sind geringfügige Voraussetzungen für den folgenden Algorithmus zur Verifikation von Worthypothesen nötig:

- Jede empfangene Worthypothese erhält fortlaufend eine eindeutige Worthypothesennummer.
- Jede empfangene Worthypothese ist vom Decoder markiert als bekannt oder neu — je nachdem, ob die Wortkopie des Decoders, auf der die Hypothese beruht, bereits verifiziert wurde oder nicht.

[5] Grundsätzlich sind auch andere Fälle denkbar, etwa derjenige Fall, bei dem ein drittes Modul die Kontrolle hat und Decoder und Parser aufruft. Solche Fälle werden hier nicht betrachtet, sondern nur die direkte Kopplung.

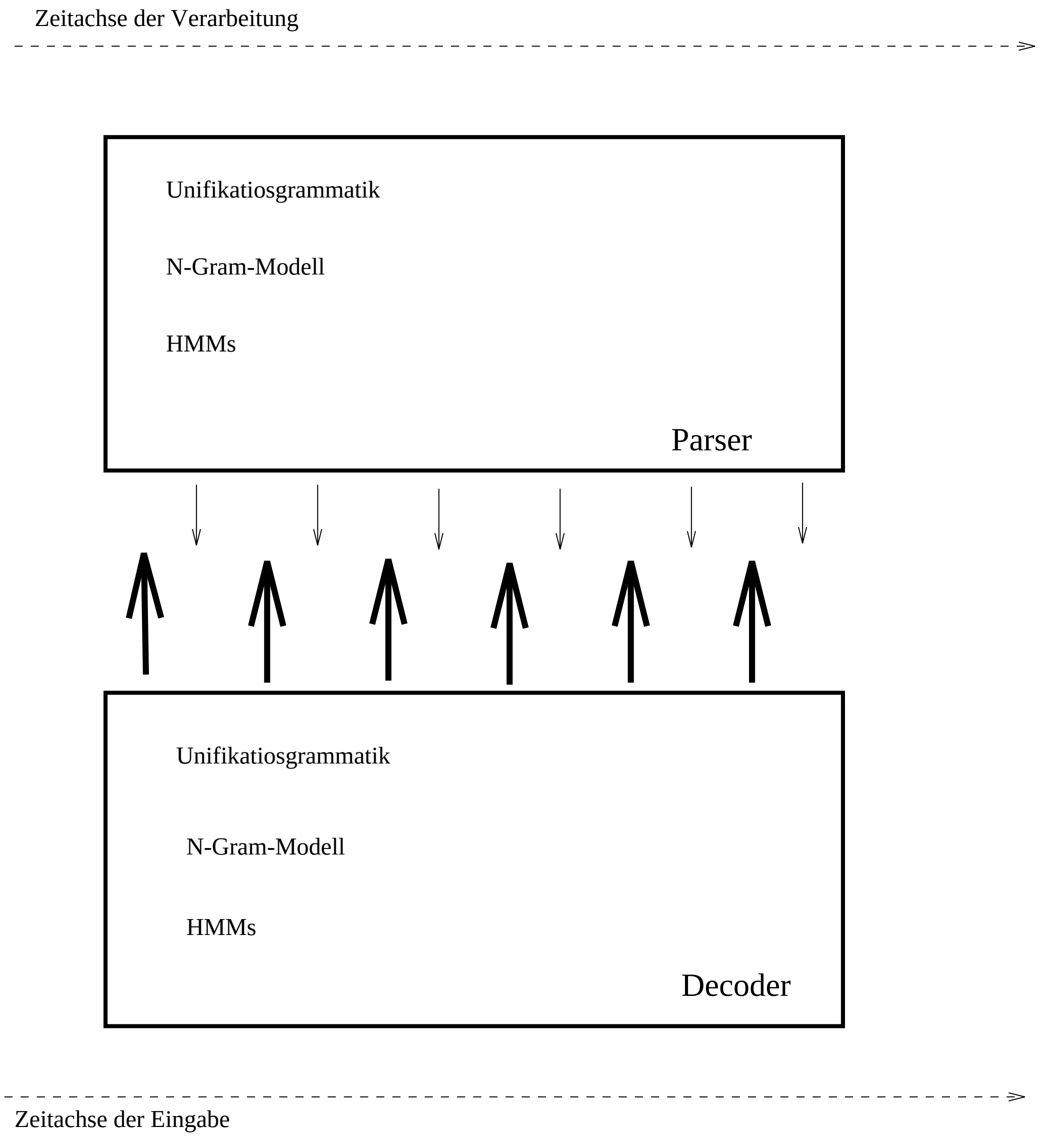

Abbildung 4.2: Parser und Decoder bei BUITDV

- Ein erweiterbarer eindimensionaler Array A wird für die Verwaltung des Verifikationsmechanismus benötigt. Für die Worthypothesennummer als Index wird in A eingetragen, ob die Hypothese noch verifiziert werden muß oder nicht[6].

Der Algorithmus für einen Parsezyklus ist dann wie folgt gegeben:

1. Für alle ankommenden Worthypothesen in einem Zyklus:
   (a) Falls die Worthypothese W_i bekannt ist, setze $A[i]$ auf 0.
   (b) Falls die Worthypothese W_i neu ist, setze $A[i]$ auf 1.
   (c) Füge W_i in die Chart ein, sodaß alle Paare von aktiven Kanten mit W_i in die Agenda eingetragen werden.
2. Bestimmte Maximum und Beamschwelle.
3. Für alle Paare (active, W_i) in PARSE-AGENDA[7], absteigend:
   (a) Parse (active, W_i)
   (b) Falls (active, W_i) erfolgreich geparst und $A[i] = 1$, dann sende VERIFY((active, W_i)). Setze $A[i]$ auf 0.
4. Für alle Paare (active, W_i) in PRUNE-AGENDA[8], absteigend:
   (a) Falls $A[i] = 1$, dann sende VERIFY((active, W_i)). Setze $A[i]$ auf 0.
   (b) Save-Agenda

Der Aufwand für den Verifikationsmechanismus ist gering. In Experimenten, die in Abschnitt 4.5 beschrieben sind, war der zusätzliche Zeitaufwand praktisch nicht meßbar.

Die Nachricht, die jeweils an den Decoder geschickt wird, ist ein Tripel, bestehend aus der Worthypothese, dem Vorgänger und der Bi-Gramm-Transition der beiden:

Definition 4.1 $VERIFY((active, W_i)) = [W_i, last(Path(active)), logP(W_i | last(Path(active)), Bi\text{-}Gramm)]$

Obwohl hier eine Bi-Gramm-Bewertung zurückgegeben wird, ist der Kontext global. Durch die Verwaltung mit dem Array A ist garantiert, daß jede Worthypothese höchstens einmal verifiziert wird. Die Verifikation oberhalb des Beams für W_i kommt aufgrund des besten parsbaren Paares (active, W_i) zustande, weil die gesamte Agenda immer sortiert gehalten wird. Die Bi-Gramm-Bewertung, die zurückgegeben wird, ist also diejenige mit dem Vorgänger aus dem global besten Suchpfad. Lokal kann es durchaus vorkommen, daß es einen besseren Vorgänger zu W_i gibt. Dieser wäre beachtet worden, wenn beim Decoding das Languagemodell direkt verwendet worden wäre. Durch den hier beschriebenen Ansatz wird jedoch der äußerungsbezogen globale N-Gram-Kontext und die Parsbarkeit mitberücksichtigt.

[6] In unserer Implementation steht dort nur 0 oder 1.
[7] ..also oberhalb des Beams.
[8] ..also unterhalb des Beams.

In einer frühen Version des Verfahrens wurden Verifikationen nur aufgrund der Paare in PARSE-AGENDA erzeugt (Schritt 3). Alle Hypothesen, deren Paare sämtlich unterhalb der Beamschwelle liegen, wurden nicht verifiziert. Der Decoder setzte dadurch solche Pfade auf eine Wahrscheinlichkeit von 0. Dies führte bei vielen Erkennungsexperimenten zu einem Zusammenbruch, sodaß gar keine überspannende Äußerung erkannt werden konnte. Indem auch Paare aus PRUNE-AGENDA verifiziert werden (Schritt 4), wird eine höhere Robustheit erzielt. Dies erklärt sich wie folgt: Für eine Wortkopie innerhalb des Decoders wird die erste Wortendehypothese oberhalb des akustischen Beams zur Verifikation an den Parser geschickt. Wenn diese nun in allen Paaren unter den Beam des Parsers fällt, stirbt die gesamte Wortkopie innerhalb des Decoders. Die Wortkopie könnte jedoch in unmittelbar folgenden Frames noch bessere akustische Bewertungen erzielen, sodaß Paare mit den entsprechenden Hypothesen im Parser wieder über die Beamschwelle gelangt wären. Es empfiehlt sich also nicht, Pfade mit Worthypothesen zu früh vollständig auf 0 zu setzen.

Indem nun – ohne daß geparst wird – der dem Pruning unterliegende Teil der Agenda nochmals durchsucht wird nach Worthypothesen, die bis dahin noch nicht verifiziert wurden, ergibt sich der folgende gewünschte Effekt. Falls eine solche Worthypothese akustisch gut war, muß die schlechte Languagewahrscheinlichkeit für den global schlechten Score verantwortlich sein. In diesem Fall enthält die Verifikationsnachricht mit hoher Wahrscheinlichkeit eine schlechte N-Gram-Bewertung. Falls jedoch nur die schlechte Akustik einen ansonsten guten Pfad hat unter den Parserbeam sinken lassen, dann enthält die Verifikationsnachricht an den Decoder möglicherweise eine relativ gute Languagebewertung. Diese wird vom Decoder schließlich auf ein Wortmodell aufaddiert, welches akustisch in folgenden Frames ebenfalls noch besser werden wird. Falls letzteres geschieht, wird in folgenden Zyklen der ganze Pfad über die Beamschwelle des Parsers gelangen. Falls nicht, ergibt sich auch kein Störeffekt.

Abbildung 4.2 stellt die globale Konfiguration von Decoder und Parser bei *Bottom Up Inkrementeller Analyse mit Top Down Verifikation* dar. Beide Module arbeiten zeitsynchron jeweils (um ein Frame versetzt) auf dem gleichen Signalabschnitt. Der Löwenanteil des Datenflusses ist bottom up. Ein vergleichsweise kleiner Datenfluss erfolgt top down. Die Module berücksichtigen bei der jeweiligen Suche alle drei Wissensquellen.

Die BUITDV Kopplung enthält ein prizipielles Problem. Während die Wortübergangsstrafe (v, w) bei BUI und auch bei TDPI (vergl. nächsten Abschnitt) beim Übergang von v nach w dem Decoder bekannt ist, steht sie bei BUITDV erst am Ende von w zur Verfügung. Frühestens wenn der korrespondierende Parse-Zyklus einen Parseschritt mit der ersten Hypothese für w ausführt, kann eine Nachricht an den Decoder die Übergangsstrafe (v, w) zur Verfügung stellen. Für den Decoder ergibt sich also keine Beschränkung für w, wenn w erst begonnen wird.

Ein ähnliches Problem ergibt sich beim Beam Decoding mit Baumlexika, wie z.B. bei Ney 1993 [66] oder Aubert et al. 1994 [6]: Bei einem Baumlexikon steht erst, wenn ein Wortendezustand – also ein Blatt des Baums – erreicht ist, fest, welche Wortform bearbeitet wurde. Erst dann kann die Initialisierung durch den Wortübergang für das Wort erfolgen, das bereits bearbeitet wurde. Ein in Aubert et. al. 1994 skizzierter Ausweg ist ein sich langsames “Anschmiegen” an den eigentlichen Übergang während der Baum durchlaufen wird. In [6] wird letztlich nur der

Unigram-Wert hierfür verwendet.

Unser Problem ist in sofern einfacher, als daß keine feinen Zwischenzustände existieren, da wir kein Baumlexikon verwenden. Bei BUITDV gibt es im Decoder die Situation, daß ein Wort begonnen wird, jedoch die Wortübergangsstrafe noch nicht bekannt ist. Zusätzlich gibt es den Zustand, daß die Wortübergangsstrafe vom Parser zur Verfügung gestellt wurde und nachträglich aufaddiert wird.

Die Bi-Gramm-Strafe des besten Vorgängers, der dem Decoder bekannt ist, stellt in unserem Fall eine gute obere Schranke für die durch den Parser zurückgegebene Wortübergangsstrafe Ver(W) eines Wortes dar. Offensichtlich kann Ver(W) nicht besser sein als die Bi-Gramm-Strafe des besten Vorgängers. Falls Ver(W) schlechter ist, wird die Differenz nachträglich – also am Ende des Wortes – aufaddiert. Wir können also durch die Bi-Gramm-Strafe ohne das Filtern durch den Parser als vorläufige Approximation der Verifikation durch den Parser einsetzen und in einem nachträglichen Korrekturschritt den "echten" Wert wiederherstellen.

In den Experimenten in Abschnitt 4.5 wurde diese Variante nicht verwendet, da der Unterschied zwischen BUI und BUITDV auch ohne diese Verbesserung gut sichtbar wird. Dennoch existiert das Problem und daher wurde eine Lösung skizziert.

BUITDV ist auch mit einem Baumlexikon verträglich. In diesem Falle wird der in Aubert et al. 1994 verwendete Korrekturschritt durch den Korrekturschritt w.o. ergänzt.[9]

4.4 Top Down Prädiktive Inkrementelle Kopplung

Während bei BUITDV der Decoder jeden Frame zuerst bearbeitet und der Parser immer einen Schritt weit hinterherhinkt, ist die *Top Down Prädiktive Inkrementelle Kopplung* genau entgegengesetzt organisiert. Bei TDPI schlägt der Parser von links nach rechts in der Sprechrichtung für jeden Zyklus Wortkandidaten vor, die mit seinem bereits bekannten linken Kontext verträglich sind. Der Decoder verwendet diese Menge von Wortformen, um in einem Frame neue Wortmodelle zu starten. Daher gilt für sämtliche Worthypothesen, die zu irgendeinem Verarbeitungszeitpunkt bottom up vom Decoder an den Parser weitergereicht werden, daß sie zu einem früheren Zeitpunkt vom Parser als *Prädiktion* vorgeschlagen worden sein müssen. Alle nicht vom Parser vorgeschlagenen Wortformen werden nicht beachtet.

Abbildung 4.3 soll darstellen, daß von links nach rechts gesehen immer abwechselnd eine größere Datenmenge vom Parser zum Decoder fließt bevor als Antwort eine kleinere Datenmenge vom Decoder als Worthypothesen an den Parser fließt.

Während die Verwendung der Prädiktionen im Decoder einfach und geradlinig realisiert werden kann[10], ist die Kalkulation der Vorhersagen im Parser rechenaufwendig und für Unifikationsgrammatiken grundsätzlich anders zu realisieren als für kontextfreie Ansätze wie etwa Päseler 1988 [67], Kita et al. 1989 [53] oder Jurafsky et al. 1994 [48].

Da es für TDPIA bereits für den kontextfreien Fall Vorbilder gibt, werden diese Ansätze kurz im Hinblick auf die Kalkulation von *Prädiktionen* durch einen Parser diskutiert. Es zeigt sich dabei, daß für Unifikationsgrammatiken eine andere

[9] Die Verträglichkeit mit Baumlexika ist eine sehr wichtige Eigenschaft, da mit linearen Lexika ein Upscaling auf große Wortschätze nicht realistisch erscheint.

[10] Nur vorhergesagte Wörter werden gestartet. Die Language-Bewertung des Parsers wird sofort aufaddiert. Vergl. hierzu Hauenstein & Weber 1994a und 1994b [39, 40].

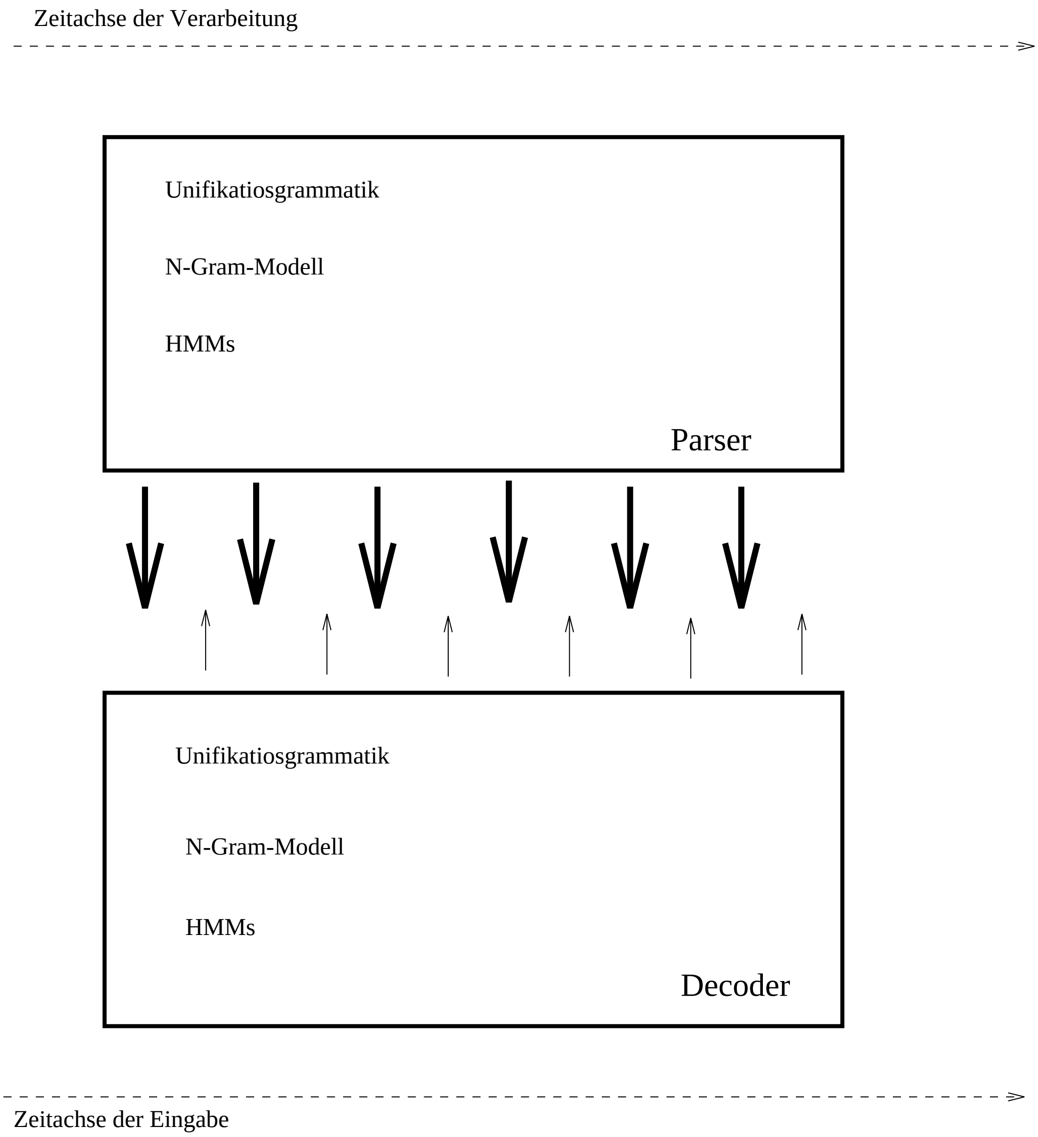

Abbildung 4.3: Parser und Decoder bei TDPI

Vorgehensweise entwickelt werden muß, da die bekannten Techniken für CFG zu ineffizient wären.

Der Ansatz von Päseler 1988 ist wiederum derjenige, der dem hier vorgestellten am ähnlichsten ist. Dort wird folgendermaßen vorgegangen:

- Es gibt kein von der Grammatik getrenntes Lexikon des Parsers. Wörter werden direkt von der Grammatik als Regeln abgeleitet. Daraus folgt, daß während der Ausführung der SEEK-DOWN-Operation — also dem Prediktorschritt im Earley–Algorithmus — die Wortebene direkt erreicht wird. In Päseler 1988 werden einfach alle Prediktoroperationen bis zur Wortebene mit der CFG ausgeführt, die sich ergebende Menge von erreichten terminalen Symbolen der Grammatik ist die *Prädiktion* für den Decoder. Eine solche Vorgehensweise ist aufgrund der teuren Unifikationsoperation prohibitiv. Die Folge wäre, daß die Kosten für das Errechnen der Prädiktion ein Vielfaches der eigentlichen Kosten des Parsing erreichen würde.

- Die Schrittweise der Kopplung ist größer bei Päseler 1988. Die Schrittweite ist in etwa Wortform–groß und nicht framebasiert wie im vorliegenden Ansatz. Die engere Verzahnung macht die Errechnung der Vorhersage auf Seiten des Parsers teurer, jedoch das Decoding billiger[11]. Wiederum spricht dies gegen eine vollständige Berechnung der *Prädiktion* innhalb unseres Ansatzes.

- Durch die ausschließliche Verwendung von akustischen Bewertungen bietet sich die vorgeschlagene Worthypothesenmenge bei Päseler als Grundlage zum Pruning an. Daraus folgt, daß diese Menge vor dem Pruningschritt auch aus diesem Grund voll expandiert werden muß. Da im LR-ACP das Pruning auf die Agenda angewendet wird, besteht hierzu keine Notwendigkeit.

Im Ansatz von Jurafsky et al. 1994 [48] wird ebenfalls eine Kopplung mit einem CFG-basierten Parser und einem Beam Decoder vorgestellt. Hierbei wird jedoch eine probabilistische CFG verwendet, sodaß, wie in unserem Ansatz dem Decoder nicht nur Wortformen, sondern auch Language-Modell-Wahrscheinlichkeiten zur Verfügung gestellt werden können. Dabei werden ebenfalls alle Analysen einer CFG ausgerechnet, um durch Aufsummieren der Ableitungswahrscheinlichkeiten die Language-Bewertung erhalten zu können.

In den folgenden beiden Abschnitten werden verschiedene Algorithmen vorgestellt, wobei deutlich wird, wo die Probleme liegen, wenn eine Prädiktion mit einem unifikationsbasierten Ansatz berechnet wird. Der letzte Algorithmus genügt dabei unseren Ansprüchen im Hinblick auf Korrektheit und Effizienz. Wiederum wird dabei die billige N-Gram-Restriktion vor der teuren unifikationsbasierten Restriktion angewendet.

4.4.1 Unifikationsbasierte Berechnung einer Prädiktion.

Mit einer frühen Version des LR-ACP sind Experimente unternommen worden zur vollständigen Berechnung von möglichen Folgewörtern in einem Wortgraphen mithilfe einer Unifikationsgrammatik. Besonderheiten ergaben sich vor allem beim

[11] Dies ist eventuell als großer Vorteil des bei Päseler vorgeschlagenen Algorithmus zu bewerten. Die von Kita vorgeschlagene prädiktive Kopplung ist ähnlich realisiert, jedoch auf Phonem- anstatt auf Wortbasis und mit einem GLRP anstatt auf Earley- bzw. ACP-Basis.

Schritt zu den terminalen Symbolen der Grammatik. Diese wurden in unseren Experimenten nicht durch den Prädiktorschritt vorhergesagt, sondern in einem Parseschritt. Hierbei kommt aber auch das Dilemma des unifikationsbasierten Parsing zum tragen. Die SEEK-DOWN Operation reicht nicht mehr aus, um alle Restriktionen anzuwenden. Um die Menge möglicher Nachfolgewörter exakt zu berechnen, muß jeweils der ganze Completer-Schritt – bekanntlich $O(n^2)$ – ausgeführt werden.

Definition 4.2 *Vollständige Kalkulation einer* Prädiktion *mit einer Unifikationsgrammatik für Knoten j.*

1. *P sei ∅. AGENDA sei leer.*
2. *ACTIVES seien alle aktiven Kanten aus j nach der letzten SEEK-DOWN Operation.*
3. *Für jedes $w \in$ LEXICON*
   (a) *Für jedes $a \in$ ACTIVES: Stoße (a, w) auf die AGENDA.*
   (b) *Parse AGENDA.*
   (c) *Falls die Startkante erreicht wird, füge w in P ein und leere die AGENDA.*

Im kontextfreien Fall reicht die SEEK-DOWN-Operation aus, um zur Wortebene zu gelangen. Bei einer Unifikationsgrammatik muß, um die gesamte zur Verfügung stehende Restriktion zu nutzen, eine möglicherweise rekursive Completer-Operation angestoßen werden. Die Gründe sind bereits aus den Abschnitten 3.2.3.4 bis 3.2.4.2 bekannt. Bei der top down Regeleinführung wird entweder ein Teil der Restriktionen vernachlässigt, der später beim Completerschritt sowieso noch wirken wird, oder aber das Parsing wird sehr teuer.

Eine verbesserte Version des obigen Algorithmus verzichtet auf den rekursiven Completerschritt bis zur Startkante der gesamten Chart. Nur die lokalen aktiven Kanten werden beachtet – also diejenigen, die im aktuellen Knoten j enden. Die Restriktion, die durch den rekursiven Schritt hinzugefügt wird, ist oft gering, weil durch die top down Operationen der in Typen kodierte *kontextfreie Anteil* und ein Teil der Merkmale propagiert werden. Nur die sogenannten *nicht lokalen Features* einer Grammatik, also jene, die Fernabhängigkeiten darstellen, werden nicht beachtet.

Definition 4.3 *Lokale Berechnung der Prädiktion mit einer Unifikationsgrammatik für den Knoten j*

1. *P sei ∅, AGENDA sei leer.*
2. *ACTIVE seien alle aktiven Kanten aus Knoten j nach der Ausführung der letzten SEEK-DOWN Operation, wobei die nichtleeren Kanten als Zielkanten ausgezeichnet werden.*
3. *Für jedes Element $w \in$ LEXICON:*
   (a) *Für jedes $a \in$ ACTIVES: Stoße (a, w) auf die AGENDA.*

(b) *Parse rekursiv nur mit Paaren (a, p), sodaß $a \in ACTIVE$.*

(c) *Falls ein Parseschritt mit einem Element aus ACTIVE gelingt, sodaß eine Zielkante erreicht wird, füge w zu P hinzu und leere die AGENDA.*

In Algorithmus 4.3 wird als Erfüllungskriterium immer noch gefordert, daß bis zum Erreichen einer nicht leeren Kante geparst wird. Das heißt, daß nur noch bezüglich eines durch die Grammatik definierten lokalen Bereichs alle Restriktionen beachtet werden.[12] Die folgende Variante der Berechnung fordert nur noch, daß für einen Kandidaten eine Operation mit einer im aktuellen Knoten endenden aktiven Kante gelingen muß. Hierbei wird die gleiche Komplexität erreicht wie in Päselers [67] Vorgehensweise für kontextfreie Grammatiken. Der Unterschied ist jedoch, daß in der kontextfreien Grammatik die terminalen Symbole durch unäre Produktionen erreicht werden, deren Verarbeitung im Prädiktorschritt miterfolgt, während solche Regeln hier nicht existieren. Durch das getrennte Lexikon, werden die auszuführenden Schritte aus den SEEK-DOWN herausgenommen und getrennt ausgeführt.

Definition 4.4 *Nichtrekursive lokale Berechnung einer Prädiktion mit einer Unifikationsgrammatik für den Knoten j.*

1. *P sei $\emptyset$, AGENDA sei leer.*
2. *Führe für einen Knoten j die SEEK-DOWN-Operation aus.*
3. *ACTIVE seien alle aktiven Kanten aus Knoten j.*
4. *Für jedes $w \in LEXICON$:*
   (a) *Für jedes $a \in ACTIVES$: Stoße (a, w) auf die AGENDA.*
   (b) *Falls ein Parseschritt gelingt, füge w zu P hinzu und leere die AGENDA.*

Bei den durchgeführten Experimenten mit 4.2 wurde eine kleine Unifikationsgrammatik mit nur 25 Regeln und 60 Worteinträgen verwendet. Dabei dauerte die Berechnung einer Prädiktion - also der ganzen Menge jeweils möglicher nächster Wortformen – zwischen 5 und 20 Sekunden. Da diese Berechnung im schlechtesten Fall für jedes Frame ausgeführt werden muß, ist eine solche Vorgehensweise nicht weiter sinnvoll. Auch die letzte Version (4.4) ist noch nicht effizient genug. Darüber hinaus wirkt hier nur noch ein kleiner Teil der Merkmale und die Beschränkung aus dem Typenanteil der Grammatik.

4.4.2 Verwendung des N-Gramm Modells zur effizienten Prädiktion

Die gewünschte Effizienz bei einer Kalkulation von Prädiktionen ist jedoch zu erreichen, wenn nicht mehr alle Lösungen berechnet werden, sondern nur noch die "guten". Die folgenden Überlegungen rechtfertigen dies:

[12] Typischerweise sind dies immer noch morphologisch ausgeprägte Eigenschaften wie Kongruenz bezüglich Numerus, Genus, Person, Beugung usw.

- Bei einer effizienten Strategie für die Einführung von leeren aktiven Kanten, wie sie in Abschnitt 3.2.3.4 bis 3.2.4.2 diskutiert wurde, muß wenigstens für den Startpunkt einer SEEK-DOWN Operation auf die Propagierung des Feature-Anteils einer Regel verzichtet werden. Zusätzlich wird auf einen Teil der durch Feature Strukturen gegebenen Beschränkung verzichtet, um auf teure Subsumptionstests für leere Kanten verzichten zu können. Um die grammatischen Operationen lokal zu halten, muß also auf eine vollständige Berücksichtigung aller durch Merkmalsstrukturen gegebenen Beschränkungen verzichtet werden.

  Eine effiziente weil lokale Strategie für die Berechnung der Prädiktion ist also bereits nur eine Approximation.

- Von den Wortformen des Lexikons, die als Kandidaten für die Prädiktion in Frage kommen, werden für die Fehlschläge die meisten Operationen benötigt. Falls eine komplette Completer-Operation pro Wortform angestoßen wird, steht ein Fehlschlag erst fest, wenn der gesamte Suchraum durchwandert wurde. Dies legt eine Strategie nahe, welche unplausible Kandidaten erst gar nicht testet.

- Aufgrund des statistischen Languagemodells besteht eine gute Möglichkeit, die Plausibilität zu bewerten, mit der eine Wortform eine bisher geparste Wortkette verlängert.

- Die nichtrekursive lokale Variante der Berechnung vernachlässigt möglicherweise lokale Beschränkungen, wie z.B. Kongruenzen innerhalb linguistischer Phrasen. Ein N-Gram-Modell berücksichtigt jedoch solche lokalen Beschränkungen in vielen Fällen.[13]

- Zur Berechnung der Prädiktion einen Suchstrahl einzuführen, paßt in das Suchkonzept des LR-ACP. Das "Pruning und Beamsearch" des LR-ACP gilt in erster Linie der Vermeidung von teuren Unifikationen.

Mit dem folgenden Algorithmus, der nicht mehr alle Lösungen produziert, ließ sich im Experiment, der Aufwand für das Errechnen einer Prädiktion in den Bereich von Zehntelsekunden drücken.[14] Dabei wird ausgenutzt, daß mithilfe des statistischen Modells Vorschläge für "gute" Nachfolger gemacht werden können, die vom Parser nur noch überprüft werden. Das Generate–and–Test Verfahren ist hier in wesentlichen Punkten überlegen. Es erlaubt insbesondere, die billige Restriktion von der teuren Restriktion anzuwenden.

Definition 4.5 *Generate–and–Test Verfahren zur effizienten Berechnung von Prädiktionen.*

1. *P' sei* $\emptyset$.

2. *NEA seien alle nicht leeren aktiven Kanten, die in j enden.*

[13] Während ein Bi-Gramm-Modell doch viele Fälle nicht abfangen kann, ist hier problemlos auch ein Tri-Gramm-Modell einsetzbar.

[14] In unserer nicht voll optimierten LISP-Implementation auf einer SUN SPARC 10.

3. *LW sei $\{lw\}$, sodaß*

   $\exists A \in NEA$ und

   $lw = last(path(A))$.

4. *Für jedes $lw \in LW$*

   Bilde $NW_i = \{w, logP_{ngram}(lw, w)\}$, für die besten N Nachfolger w von lw, sodaß $logP_{ngram}(lw, w)$ maximal wird.

5. *Füge jeden vorgeschlagenen Nachfolger w einmal in P' ein.*

6. *P sei $\emptyset$. AGENDA sei leer.*

7. *Führe für einen Knoten j die SEEK-DOWN-Operation aus.*

8. *ACTIVE seien alle aktiven Kanten aus Knoten j.*

9. *Für jedes $w \in P'$:*

   (a) *Für jedes $a \in ACTIVES$: Stoße (a, w) auf die AGENDA, sodaß $score(a, w) = Normalize(Denormalize(score(a)) + logP_{ngram}(lw, w))$, wobei $lw = last(path(a))$*

   (b) *Prune*

   (c) *Falls ein Parseschritt gelingt, füge $(w, score)$ zu P hinzu, wobei score der aktuelle N-gram-Übergang nach w beim ersten erfolgreichen Pars ist, und leere die AGENDA.*

Um mithilfe des statistischen Languagemodells Vorschläge für die Verlängerung von bereits geparsten Wortketten bis zu Knoten j zu erhalten, werden zuerst die letzten Worthypothesen benötigt, die im Knoten j enden. Für jede solche Hypothese muß wenigstens eine aktive Kante existieren, die sie enthält. Falls nicht, sind alle Pfade mit dieser Hypothese bereits unter den Suchstrahl gefallen oder waren nicht parsbar.

In Schritt 4 kann N auch die Größe des Lexikons sein. Durch den späteren Pruneschritt verliert dies an Gewicht, solange N nicht zu klein gewählt wird. Die Haupteigenschaft dieses Schrittes ist, daß er keine Rechenzeit benötigt. Für ein gegebenes Bi-Gramm-Modell und N kann eine Liste von besten Nachfolgern für jede Wortform vorherberechnet werden.[15]

Wie bereits bei BUITDV ist die Bewertung, die mit einer Wortform in P verbunden ist, nicht das lokale Maximum eines Übergangs von irgendeinem Vorgänger. Der Einfluß des bereits geparsten linken Kontextes ist hier sehr stark. Der N-Gram-Wert, der mit einem w in der Prädiktion verknüpft ist, ist der N-Gram-Anteil des letzten Übergangs des besten Scores eines Pfades mit w verlängert. Wenn die Pädiktion an den Decoder übergeben wird, sodaß dieser nur die Modelle neu startet, deren

[15] In unserer Implementation wurden Hashtabellen verwendet, deren Schlüssel die lw und Werte Listen von Zeigern auf die w sind.

Wortformen in P enthalten sind, dann erhält er somit nicht lokale, sondern global ausgewählte lokale Information.

In Algorithmus 4.5 ist in Schritt 9 absichtlich auf eine detaillierte Spezifikation des Erfüllungskriteriums und der Details der Auswahl von Paaren auf der Agenda verzichtet worden. An dieser Stelle ist Algorithmus 4.5 verträglich mit allen drei in Abschnitt 4.4.1 vorgestellten Verfahren.

Die Komplexität des Algorithmus ist abhängig von Schritt 9 und der Anzahl der Wortformen, für die Schritt 9 ausgeführt werden muß. Das bedeutet, daß das Verfahren mit größeren Wortschätzen teurer wird, solange ein wortformbasiertes Modell verwendet wird. Durch ein Klassenmodell kann dies jedoch umgangen werden. Allerdings müssen die Klassen des N-Gram-Modells in diesem Fall mit Klassen von Merkmalsstrukturen im Lexikon der Unifikationsgrammatik zusammenfallen. Für eine Klasse des N-Grams muß eine Merkmalsstruktur existieren, die alle Merkmalsstrukturen derjenigen Wortformen im Lexikon subsummiert, die zu dieser Klasse gehören. Dabei treten zwei Schwierigkeiten auf: Zum einen sind Bi-Gramme wie bei Kneser & Ney 1991 [54], die mit Clustering–Verfahren erstellt werden, sodaß die Klassen über einer Lernstichprobe optimiert sind, nicht mehr einsetzbar. Zum anderen dürfen die Klassen nicht zu groß sein, weil dann die prototypischen Merkmalsstrukturen der Klassen sehr allgemein werden. Dies würde dazu führen, daß die Restriktionen aus der Unifikationsgrammatik zunehmend wirkungslos werden.

In unseren Experimenten wurde nur ein Vokabular von 365 Wortformen verwendet. In diesem Bereich ist die wortformbasierte Vorgehensweise völlig unproblematisch. Wir verwendeten in Schritt 9 den Algorithmus 4.3. Ambiguitäten im Lexikon der Unifikationsgrammatik wurden so behandelt, daß für jedes w in Schritt 9 mehrere Testkanten mit den jeweiligen Merkmalsstrukturen eingesetzt wurden.

4.5 Experimente mit drei LR-inkrementellen Kopplungen

Mit den drei LR-inkrementellen Kopplungen wurden Experimente unternommen, wobei die verwendeten Wissensbasen[16] konstant gehalten wurden. Auch die Beambreiten für Parser und Decoder waren jeweils gleich im ersten Experiment 4.4 und im zweiten Experiment 4.5.[17].

Für alle drei Varianten für eine Kopplung wurde zweimal das Verhalten über 10 Testsätze gemessen. Einmal mit engen Suchstrahlen für beide Komponenten und einmal mit weiten Suchstrahlen.

Eine Äußerung konnte nur erkannt werden, wenn die richtige Kette in der Menge der bottom up übertragenen Worthypothesen wirklich enthalten war. Die verwendeten akustischen Modelle waren daher so adaptiert, daß sie 90% Worterkennungsrate erbrachten.

Die Testsatzperplexität des Wort-Bi-Gramms wurde mit 54.28 gemessen. Die

[16] Dies sind Unifikationsgrammatik, Bi-Gramm-Modell und HMMs.

[17] Die Werte für den Parser waren 0.5 im ersten Experiment und 0.3 im zweiten Experiment. Die Logarithmen dieser Werte wurden auf das Maximum (eigentlich Minimum, aber bester Wert) aufaddiert und ergaben die Beamschwelle. Der Decoder benutzte bei weitem Beam einen Offset von 10e-8 und bei engem Beam eine Strafe von 10e-6. Die unterschiedlichen Größenordnungen kommen, da die Scores im Decoder absolut und im Parser normalisiert sind.

Mode	Time	BU Hypos	TD Hypos	Rec.Rate	Edges	Gridpoints
AC	47.4	–	–	0.5	–	790
BUI	96.9	385	–	0.7	6195	790
BUITDV	58.2	289	41	0.5	5827	1045
TDPI	220.5	223	7621	0.7	4954	78

Abbildung 4.4: Experiment1: Enge Suchstrahlen.

Mode	Time	BU Hypos	TD Hypos	Rec.Rate	Edges	Gridpoints
BUI	2911.7	1465	–	0.7	19769	1712
BUITDV	115.8	776	135	0.7	9765	2047
TDPI	282.7	415	8304	1.0	7519	184

Abbildung 4.5: Experiment2: Weite Suchstrahlen.

Unifikationsgrammatik enthält 31 Regelschemata und das Lexikon beschreibt 365 Vollformen mit einer durchschnittlichen Ambiguität von 1.3.

Bei allen drei Experimenten wurden Parser und Decoder als je ein Prozeß auf einem eigenen Prozessor realisiert. Decoder und Parser starteten immer gleichzeitig.

Die Meßwerte in den folgenden Tabellen ergeben sich wie folgt:

Time: Gemessene Zeit des Parserprozesses bis zur ersten gefundenen Analyse.

BU-Hypos: Anzahl der während eines Testlaufs vom Decoder an den Parser gesendeten Worthypothesen.

TD-Hypos: Anzahl der während eines Testlaufs vom Parser an den Decoder gesendeten Worthypothesen.

Rec.Rate: Satzerkennungsrate. Gezählt wurden nur die korrekten Sätze mit 1 — alle anderen mit 0.

Edges: Anzahl der beim Parsing erzeugten Chartkanten. Kanten, die bei TDPI zur Berechnung von Prädiktionen erzeugt wurden, wurden nicht mitgezählt.

Gridpoints: Maximum der Anzahl konkurrierender Modellzustände des Decoders während eines Testlaufs

Alle Werte sind über alle Testsätze gemittelt. Die Gegenüberstellung von einmal engen und einmal weiten Suchstrahlen beider Module soll einen Eindruck vermitteln, wie sich das Verhalten der Kopplungen bei verschiedenen Anforderungen entwickelt. Die Mischung von engem und weitem Strahl erschien uns relativ sinnlos. Für ein gegebenes System in der Praxis muß das Verhältnis der Strahlen ebenso wie die Gewichte der Modelle sowieso eingeregelt werden. In dieser Arbeit ging es mehr um prinzipielles Verhalten. Dies ist aus den Experimenten gut ablesbar.

Die augenfälligste Beobachtung, die daher zuerst Beachtung findet, ist der enorme Anstieg der durchschnittlichen Laufzeit der BUI-Kopplung von Experiment 1 zu Experiment 2. Die Höhe des Sprungs erklärt sich jedoch dadurch, daß bei BUI und weitem Suchstrahl in einigen Fällen der Hauptspeicher von 64 MB der Maschine überschritten wurde, auf welcher der Parsingprozess verarbeitet wurde. Dadurch

Mode	Time	BU Hypos	TD Hypos	Edges	Gridpoints
BUI	30.0	3.8	-	3.2	2.2
BUITDV	2.0	2.7	3.3	1.7	2.0
TDPI	1.3	1.9	1.1	1.5	2.4

Abbildung 4.6: Anstieg der Werte von engem zu weitem Beam.

wurde der Prozess zeitweise ausgelagert, was den absoluten Zeitbedarf enorm ansteigen ließ. Die Anzahl der im Verlauf des Parsing erzeugten Kanten gibt jedoch ein realistisches Bild des Verarbeitungsaufwands, in gewissem Sinne auch die Anzahl der bottom up übertragenen Worthypothesen. Die Tabelle 4.6 zeigt daher noch einmal den Anstieg dieser drei Werte von Experiment 1 zu Experiment 2.

BUI verhält sich hier deutlich schlechter als die beiden top down Kopplungen. Außerdem ist das Verhalten von TDPIA bezüglich des Anstiegs der drei Werte signifikant besser als das Verhalten von BUITDV.

Von großer Bedeutung im Hinblick auf den LR-ACP ist die Beobachtung, daß in allen drei Kopplungen die Anzahl der erzeugten Kanten im Experiment in etwa der gleichen Größenordnung ansteigt wie die Anzahl der gesendeten Worthypothesen. Dies ist eine Eigenschaft des Beamsearch. Wenn ohne Pruning alle Kanten erzeugt würden, wäre eine kombinatorische Explosion zu beobachten.

Mit weiterem Beam wird das Verhältnis sogar etwas besser. Indem aufgrund des weiteren Beams auch schlechter bewertete Pfade weiterverfolgt werden, steigt der Anteil derjenigen Pfade an, die nicht von der Unifikationsgrammatik akzeptiert werden.

Durch dieses "quasi-lineare" Verhalten im experimentierten Bereich wird die entscheidende Größe die Anzahl der bottom up übertragenen Hypothesen. Zunächst unterscheidet die Kopplungen also, ob – und wie stark – durch die top down Information die bottom up gesendeten Hypothesen beschränkt werden. In dieser Hinsicht ist die Beschränkung durch TDPI stärker als durch BUITDV. Für TDPI ist dies auch an den offenen Markovzuständen abzulesen. Das dies für BUITDV nicht gilt, hängt mit dem nachträglichen Aufaddieren der Language-Strafe zusammen, welches in Abschnitt 4.3 bereits diskutiert wurde.

Neben der Reduktion der bottom up Hypothesen durch top down Mechanismen sind die Kosten hierfür der zweite entscheidende Faktor. Hierbei ist die BUITDV Kopplung deutlich besser. An den absoluten Laufzeiten kann abgelesen werden, daß TDPI für weniger Hypothesen und weniger Kanten mehr Zeit benötigt als BUITDV.

Dennoch ist das Verhältnis nicht so klar, wie es zunächst den Anschein hat. Tabelle 4.6 zeigt u.a. die Werte für die Anstiege der top down Hypothesen. Dies ist in etwa ein Maß für den Aufwand der Berechnung der top down Information. Dieser Wert ist bei TDPI mit 1.1 sehr klein und ein Beleg dafür, daß trotz der weiteren Suchstrahlen von Parser und Decoder in Experiment 2 die Hauptquelle der Beschränkung weitgehend konstant blieb. Dem scheint zwar der relativ große Anstieg der Markovzustände in Decoder bei TDPI zu wiedersprechen. Die entsprechenden absoluten Werte liegen jedoch generell eine Größenordnung unter denen von BUI und BUITDV.

Die Laufzeit des Parsers zerfällt bei TDPI in den Anteil des eigentlichen Parsing und den Anteil für die Berechnung der Prädiktion. Letzterer wird offensichtlich

Mode	Edges/TDH 1	Edges/TDH 2	2/1	Edges/Time 2/1
TDPI	1.5	1.1	0.73	0.84

Abbildung 4.7: Top Down Hypothesen und Kanten, Top Down Hypothesen und Zeit bei TDPI

Mode	Narrow Beam	Wide Beam
BUI	16.1	13.5
BUITDV	20.2	12.6
TDPI	22.2	18.1

Abbildung 4.8: Die Verhältnisse von bottom up Hypothesen zu erzeugten Kanten.

stärker von der Größe des Lexikons beeinflusst als von der Menge der bottom up erhaltenen Hypothesen und steigt auch bei weiterem Parsbeam nicht wesentlich an. Das Verhältnis von Kanten zu top down Messages unterstreicht dies.

Im Verhältnis zu BUITDV ist der Overhead für die Kalkulation von Prädiktionen bei TDPI deutlich messbar. Obwohl die Menge wirklich zu parsender bottom up Worthypothesen kleiner ist, ist die absolute Laufzeit größer. Der Overhead steigt jedoch wesentlich langsamer als die eigentliche Arbeit für das Parsing. Während das Verhältnis der Laufzeiten von BUITDV und TDPI bei engem Beam noch 3.8 war, entstand bei weitem Beam nur noch ein Verhältnis von 2.4.

Dies ist ein sehr gutes Ergebnis für die TDPI Kopplung, denn die Prädiktion wurde in allen Tests für jedes Frame von 10ms ausgerechnet. Hierbei wurde der Großteil der Kalkulationen mehrfach ausgeführt, da sich die Worthypothesen in einem solchen Bereich oft kaum oder gar nicht änderten. Ein Beispiel hierfür ist der Anfangsbereich einer Äußerung. In allen Testsätzen gibt es innerhalb der ersten halben Sekunde nur Worthypothesen für Stille. Der Parser muß bei TDPI folglich in diesem Bereich fünfzig mal die selbe Kalkulation mit dem gleichen Ergebnis ausführen. Dies bedeutet, daß die Möglichkeiten für TDPI keinesfalls ausgeschöpft sind. Die guten Satzerkennungsraten von TDPI schließlich legen es nahe, diese Kopplung noch näher zu betrachten im Hinblick auf die Eingrenzung des Overheads[18].

Neben den entscheidenden Faktoren der Beschränkung und der diesbezüglichen Kosten ist die Qualität der übertragenen bottom up Worthypothesen der dritte wichtige Faktor. Um eine Abschätzung hierfür zur erhalten, betrachten wir die Tabelle 4.8. Hier sind jeweils für alle Kopplungen und Experimente das Verhältnis von Worthypothesen zu Kanten gegeben. Die absolute Größenordnung, in der sich die Werte bewegen, ist im übrigen von der Größe der Grammatik abhängig. Dies liegt daran, daß hier nur ein N-Gram und nicht die Grammatischen Ableitungen in das Pruning hineingerechnet sind.[19]

Zu erwarten war, daß bei TDPI die meisten Kanten pro Hypothese entstehen würden. Bei dieser Kopplung wählt ja der Decoder nur zwischen möglichen Verlängerungen von bereits geparsten Pfaden aus. Jede mögliche bottom up Hypothese muß also parsbar sein, es sei denn sie fällt unter die Beamschwelle des Parsers.

[18] Siehe hierzu Abschnitt 4.6.4.
[19] Siehe hierzu auch Abschnitt 5.

Die durchweg hohe Erkennungsrate korrespondiert hiermit. Die TDPI-Kopplung produziert immer einen erkannten Satz, auch wenn dies nicht der wirklich gesprochene ist. Bei engem Beam ist die Erwartung relativ hoch. Es werden also teilweise völlig unsinnige, aber syntaktisch korrekte Sätze erkannt. Bei weitem Beam ist das Verfahren robust.

Während bei engem Beam die Qualität der bottom up Hypothesen relativ gut ist, erreicht BUITDV bei weitem Beam den schlechtesten Wert überhaupt. Die Erklärung hierfür liegt in der Art, in der der Decoder die vom Parser erhaltene Language-Strafe verwendet. Gleichzeitig ist dies auch die Erklärung für die hohen Maxima von BUITDV bei den konkurrierenden Markovzuständen.

Als vorläufiges Fazit ergibt sich:

- Der Arbeitsaufwand steigt im experimentierten Bereich beim LR-ACP linear zur Anzahl der bottom up Hypothesen.

- Top down Kopplungen sind bei links rechts inkrementeller Vorgehensweise nicht nur möglich; sie wirken sich auch günstig aus in Bezug auf Erkennung und Effizienz.

- Eine radikalere top down Beschränkung ist mit einem Overhead verbunden. Eine schwächere top down Beschränkung ist ohne Overhead realisierbar, führt jedoch nur zu besserer Effizienz und nicht zu besserer Qualität.

Eine Erklärung für die Beobachtung liegt auf der Hand. Bekanntlich ist die Erkennungsrate eine Funktion der Perplexität. Folglich muß die TDPI-Kopplung die Perplexität des Bi-Gramms verändern. Dies ist auch plausibel, denn bei der TDPI-Kopplung werden nur die durch eine Prädiktion getriggerten Wortmodelle vom Decoder gestartet. Dies entspricht der Bi-Gramm-Wahrscheinlichkeit 0 für alle nicht in der Prädiktion enthaltenen Wortformen.

4.6 Abstraktion von Zeitpunkten zu Knoten

4.6.1 Linksverbundene Wortgraphen und Familien von Worthypothesen

Im vorangegangenen Abschnitt wurde Bezug genommen auf bottom up Worthypothesen und Gruppen von Worthypothesen, die sich auf das selbe aktive Modell des Decoders beziehen. In diesem Abschnitt wird hierauf näher eingegangen, da es sich um typische Phänomene handelt, die bei LR-Decoding und LR-Parsing und im besonderen bei der LR-inkrementellen Kopplung auftreten.

In Abschnitt 2.3.3 wurden drei Definitionen für Schnittstellen angeboten, die üblicherweise in nicht LR-inkrementellen Systemen verwendet werden. Dies waren die N-beste-Schnittstelle, der verbundene Wortgraph und die Lattice.

Zweifellos bilden die in den drei LR-inkrementellen Kopplungen übertragenen bottom up Worthypothesen – a posteriori betrachtet – eine Lattice. Denn nach der Definition in 2.3.3 ist jede Menge von Worthypothesen über einem festen Zeitraster eine Lattice.

Desweiteren ist offensichtlich, daß es sich nicht um einen verbundenen Wortgraphen handelt. Es ist durchaus möglich – und kam auch in den Experimenten vor

– daß ein Knoten zwar der Endeknoten einer Hypothese war, von diesem Knoten aber keine weitere Hypothese startete. Die Verbundenheitsbedingung gilt also nicht nach rechts. Die Ursache hierfür liegt an der Kombination zweier LR-inkrementeller Verfahren, die schritthaltend arbeiten.

Bei der LR-inkrementellen Kopplung werden Worthypothesen sofort übertragen, sowie der Decoder für ein Frame Wortendezustände erreicht, die oberhalb des Beams liegen. Zu diesem Zeitpunkt ist unbekannt, ob im nächsten Frame beginnende Wortkopien über dem Beam des Decoders liegen werden oder nicht. Auch eine Verzögerung des Sendens von Worthypothesen kann dies nicht prinzipiell verhindern, da ein Pfad von Worthypothesen auch erst mehrere Worthypothesen später keinen Nachfolger mehr haben kann.[20]

In einer Standardarchitektur – wenn also Decoder und Parser nicht LR-inkrementell gekoppelt sind – kann dies verhindert werden, indem ein Rückwärtsschritt vom Äußerungsende aus nur die wirklich verbundenen Hypothesen aufsammelt und erst dann an den Parser sendet, wie im original Viterbi Algorithmus.

Die Linksverbundenheit ist jedoch bei LR-inkrementeller Kopplung weiterhin gegeben und ist eine inhärente Eigenschaft sowohl des Beamdecoders als auch des LR-ACP. Nur bei Wortendehypothesen im Decoder werden neue Wortkopien angestoßen. Und bereits das standard Earley-Verfahren setzt nur aktive Kanten ein, wo geparste Wörter enden, sodaß auch nur dort nachfolgende Wörter in das Parsing einbezogen werden können.[21]

Die Schnittstelle zwischen Decoder und Parser bei LR-inkrementeller Kopplung ist also ein linksverbundener Wortgraph:

Definition 4.6 *Linksverbundener Wortgraph (LCW)*

> *Ein linksverbundener Wortgraph W ist eine Menge von konkurrierenden Worthypothesen, sodaß für jede Worthypothese $w_j \in W$ wenigstens ein Vorgänger w_i existiert, sodaß der Endezeitpunkt von w_i der Anfangszeitpunkt von w_j ist oder w_j hat den Anfangszeitpunkt 0.*

Bemerkung: *Ein linksverbundener Wortgraph ist ein gerichteter azyklischer Wurzelgraph, über dessen Knoten eine totale Ordnung definiert ist.*

Aus dieser Definition und den Definitionen 2.14 und 2.16 folgt $WG \subset LCW \subset L$.

Wenn ein linksverbundener Wortgraph von links nach rechts geparst wird, kann nicht prinzipiell verhindert werden, daß eventuell auch Sackgassen mitverarbeitet werden. Die Möglichkeit einer Parallelisierung von Worterkennung und Parsing und die effektive Ausnutzung von top down Beschränkung haben hier ihren Preis. Dennoch ist für viele Fälle eine Reduktion solcher Sackgassen möglich, ohne daß eine enge LR-inkrementelle Kopplung aufgegeben wird – also ohne Einsatz eines Lookahead oder ähnlicher Techniken. Im nächsten Abschnitt 4.6.4 wird ein Verfahren vorgestellt, daß LCWs stark reduzieren kann, ohne korrekte Pfade zu verlieren.

[20] Ein Lookahead – also hier eine Verzögerung des Sendens – kann natürlich die Menge von Sackgassen verkleinern, eben genau um diejenigen Sackgassen, deren Länge die Länge des Lookahead-Fensters nicht überschreitet.

[21] Das standard Earley-Verfahren verwendet einen top down Prädiktorschritt. Ein bottom up Prädiktorschritt, wie er in der Chart möglich ist, kann jedoch bei LR-inkrementellem Einlesen von Worthypothesen Sackgassen ebenfalls nicht vermeiden. Zusätzlich fällt hier jedoch auch die Verbundenheitsbedingung von links weg.

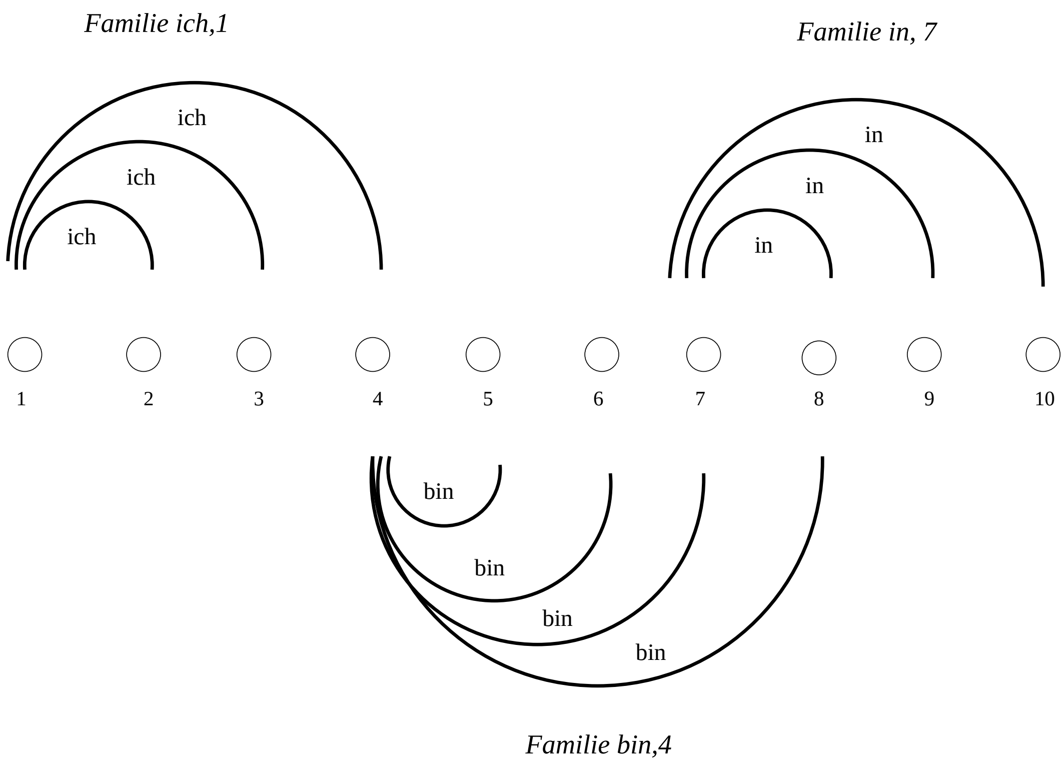

Abbildung 4.9: Familien von Kanten.

Ein großer Anteil der Sackgassen in den LCW, die während der Experimente auftraten, enden in Worthypothesen, für die eine ansonsten identische Hypothese im Vorgänger- oder Nachfolgeknoten endet.

Die typische Situation ist in Abbildung 4.9 dargestellt.

Solche Gruppen von Hypothesen entstehen, wenn beim Beamdecoding der Wortendezustand einer Wortkopie mehrere Frames lang oberhalb des Beams des Decoders liegt. Dieser Fall ist der Normalfall, denn wenn der Suchstrahl des Decoders so eng wäre, daß dies verhindert werden könnte, würde die akustische Erkennung zu schlecht werden. Normalerweise steigt für eine Wortform der Endezustand kontinuierlich an, erreicht sein Maximum, und sinkt wieder ab. Dabei ist der Endezustand eine Folge von Frames lang oberhalb des Suchstrahls und entsprechend werden Worthypothesen gebildet. In Abbildung 4.10 ist ein solcher Verlauf dargestellt.

Eine solche Gruppe von Worthypothesen aus der gleichen Wortkopie nennen wir eine Familie von Worthypothesen. Entsprechend definieren wir in der Chart eine Familie von Kanten.

Definition 4.7 *Eine Menge von terminalen Kanten heißt* Familie F*, falls alle* $f \in F$ *den selben Startzeitpunkt und den selben Lexikoneintrag besitzen.*

Die obige Definition ist so gewählt, daß im Falle einer lexikalischen Ambiguität in der Unifikationsgrammatik eine Wortkopie des Decoders zu mehreren Familien von Kanten in der Chart führen kann.

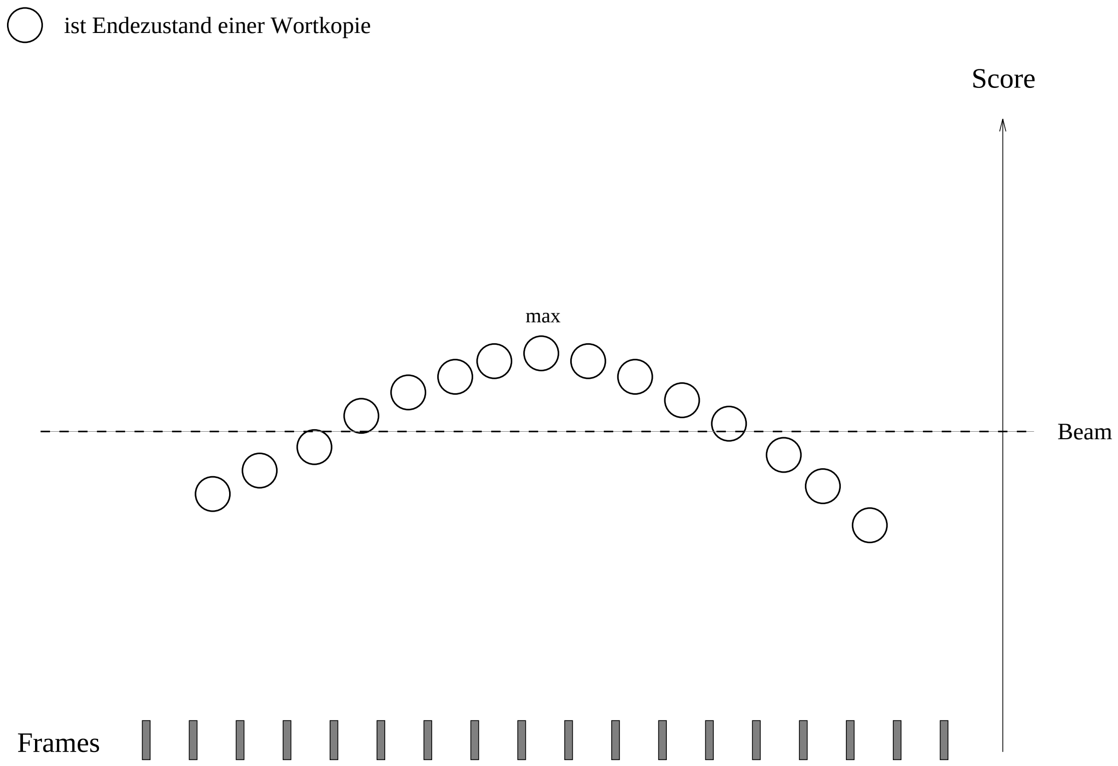

Abbildung 4.10: Verhalten eines Wortendezustands mit Beam.

In Abbildung 4.9 wird deutlich, daß nicht alle Mitglieder einer Familie eine Fortsetzung haben. Da jedes Mitglied einer Familie die gleiche Merkmalsstruktur besitzt, wird natürlich möglicherweise beim Parsing die gleiche Operation mehrfach ausgeführt. Dies ist auch der Fall, wenn ein Decoder mit Wortkopien [22] verwendet wird. Von den ausgeführten Operationen wiederum münden eventuell einige in Sackgassen.

Das Verhältnis von Familien zu bottom up Worthypothesen wird sehr gut in den Meßwerten von BUITDV sichtbar. Da, wie in Abschnitt 4.3 erläutert, bei BUITDV jede Wortkopie des Decoders nur eine top down Verifikation des Parsers auslöst, korrespondiert die Anzahl der top down Worthypothesen bei BUITDV in den Tabellen 4.4 und 4.5 mit der Anzahl der auftretenden Familien, wobei jedoch lexikalische Ambiguitäten der Unifikationsgrammatik hier nicht zu mehreren Familien führt.

Der Wert für BUITDV von 776 bottom up Hypothesen zu nur 135 top down Hypothesen in Experiment 2 zeigt, wie stark die Familien durchschnittlich verzweigen. Der folgende Abschnitt widmet sich der Abbildung verschiedener Wortendehypothesen zu einer Familie von Kanten und der Vermeidung von Doppelarbeit.

Die Verbundenheitsrelation für Familien spielt im folgenden eine wichtige Rolle und wird wie folgt definiert:

Definition 4.8 *Verbundenheit von Familien.*

> *Eine Familie F ist Vorgänger einer Familie G, wenn es $f \in F$ und $g \in G$ gibt, sodaß $end(f) = begin(g)$.*

4.6.2 Effekt von Familien für die Parsingkomplexität

Bei den Experimenten in Abschnitt 4.5 wurden alle Mitglieder einer Familie von Kanten wie verschiedene Kanten behandelt. Bei der LR-inkrementellen Kopplung wurden alle vom Beamdecoder gefundenen Worthypothesen sofort an den Parser geschickt, ohne eine Zusammenfassung vorzunehmen. Dies führt beim Parsing zu folgendem Effekt. Für die erste Kante f_1 einer Familie F werden alle aktiven Vorgängerkanten $a \in$ ACTIVE mit f_1 auf die Agenda geworfen und die entsprechende Completer-Operation wird ausgelöst. Die dabei entstehenden Kanten nennen wir Successors(f_1). Desgleichen werden $i-1$ Frames später für f_i entsprechende Operationen ausgeführt.

Es gilt $\mathrm{cat}(f_1) = \mathrm{cat}(f_i)$, da per Definition alle Mitglieder einer Familie die selbe Merkmalsstruktur besitzen.

Da für alle $f \in F$ der Anfangsknoten gleich ist, entsteht für jedes Paar (a, f_1) auf der initialen Agenda von Zyklus k ein Paar (a, f_i) auf der Agenda $k + (i - 1)$, sodaß $a \in$ ACTIVE.

Da eine Completeroperation nur von den $a \in$ ACTIVE und von den Merkmalsstrukturen der Worthypothesen abhängt, muß gelten, daß für jede Kante aus Successors(f_1) genau eine Kante in Successors(f_i) existiert, die sich nur in den Endpunkten unterscheiden und umgekehrt. Die beiden Kanten führen also zu identischen Folgeoperationen und Folgekanten, bis auf die Endpunkte aller neu entstehenden Kanten.

[22] Vergl. u.a. Ney 1993, [66].

Für eine gegebenen Familie mit k Kanten ist also der Aufwand für das Parsing k mal so groß wie für ein einziges Familienmitglied. Die mittlere Länge von Familien ist also eine gute Abschätzung für den Overhead, der durch das Phänomen entsteht.

Entsprechend steigt im Falle der TDPI-Kopplung der Aufwand für die Berechnung von Vorhersagen an. Die Strahlensuche, die ebenfalls Auswirkungen für die Bearbeitung von Familien hat, vernachlässigen wir in diesem Abschnitt. Auf die damit zusammenhängenden Effekte und deren Beherrschung wird in Abschnitt 4.6.4 ausführlich eingegangen.

Das Problem bei der Vermeidung von Doppelarbeit liegt nun darin, daß die Reihenfolgebeziehungen nicht verändert werden sollen.

4.6.3 Bisherige Ansätze

Das Problem, Familien von Worthypothesen nur einmal darzustellen, jedoch die Verbundenheitsrelationen zwischen Worthypothesen nicht zu verändern bezeichnen wir im folgenden als *Time Mapping*. Es wurde in der Literatur bisher auf zwei Arten behandelt. Zusammenfassung durch heuristische Kriterien, wie bei Chien et al. 1990 (1993) [16, 17] und durch eine starre Abbildung von Zeiten zu Knoten wie bei Weber 1992 [95].

In Chien et al. 1990 bzw. Chien et al. 1993 [16, 17] ist ein Verfahren dargestellt, das für nicht verbundene Lattices eine Abbildung auf Chartknoten vornimmt, sodaß in der Chart ein verbundener Wortgraph entsteht. Für die Eingabe (die Lattice) wird angenommen, daß Familien nur durch eine Worthypothese dargestellt werden. Hingegen gibt es Überlappungen und Lücken, die durch das Verfahren aufgelöst werden.[23]

Angewendet auf unser Problem würde eine solche Lattice entstehen, wenn z.B. für jede Familie F nur f_1 oder $f_{\max}$[24] ausgewählt würde.

Der erste Teil des Verfahrens von Chien et al. 1990 bzw. 1993 besteht in der Zusammenfassung von Knoten nach einer heuristischen Regel, die leider in beiden Veröffentlichungen nicht eindeutig erklärt ist.[25]

Die interessante Idee von Chien et al. 1990, die zu einem *Time Mapping* verwendet werden kann, ist die Einführung sogenannter Sprungkanten – der zweite Schritt ihres Verfahrens.

Um die Verbundenheit zweier Worthypothesen herzustellen, die keinen gemeinsamen Knoten teilen, wird eine zusätzliche Sprungkante eingeführt. Die Fundamentale Regel des ACP kombiniert dann nicht nur direkt benachbarte Kanten, sondern solche, die entweder benachbart sind, oder aber durch eine Sprungkante verbunden.

Ein solches Verfahren könnte also wie folgt aussehen:

[23] Wie diese Lattices erzeugt werden, wird in [16, 17] nicht näher erläutert. Insofern ist die Behauptung, die Abbildung erhalte die Verbundenheitsrelationen, nicht überprüfbar.

[24] $f_{\max}$ sei das $f_i \in F$ mit dem besten akustischen Score.

[25] In [17] sind die Details dieser Abbildung offen gelassen und es wird auf [16] verwiesen. Dort hingegen ist das Verfahren so nebulös dargestellt, daß verschiedene Interpretationen möglich sind. Unserer Versuche mit verschiedenen Möglichkeiten der Rekonstruktion des Verfahrens führten entweder zu keiner Ersparnis an Knoten, oder aber zu einer so starken Reduktion, daß ein Zusammenbruch der Reihenfolgebeziehungen beobachtet werden konnte, in dem Sinne, daß sehr viele zusätzliche zu parsende Pfade entstanden, auch für weit voneinander entfernte Worthypothesen. Es muß jedoch festgehalten werden, daß Chien et al. von speziellen Lattices ausgegangen sein müssen, über deren Details wir nichts wissen.

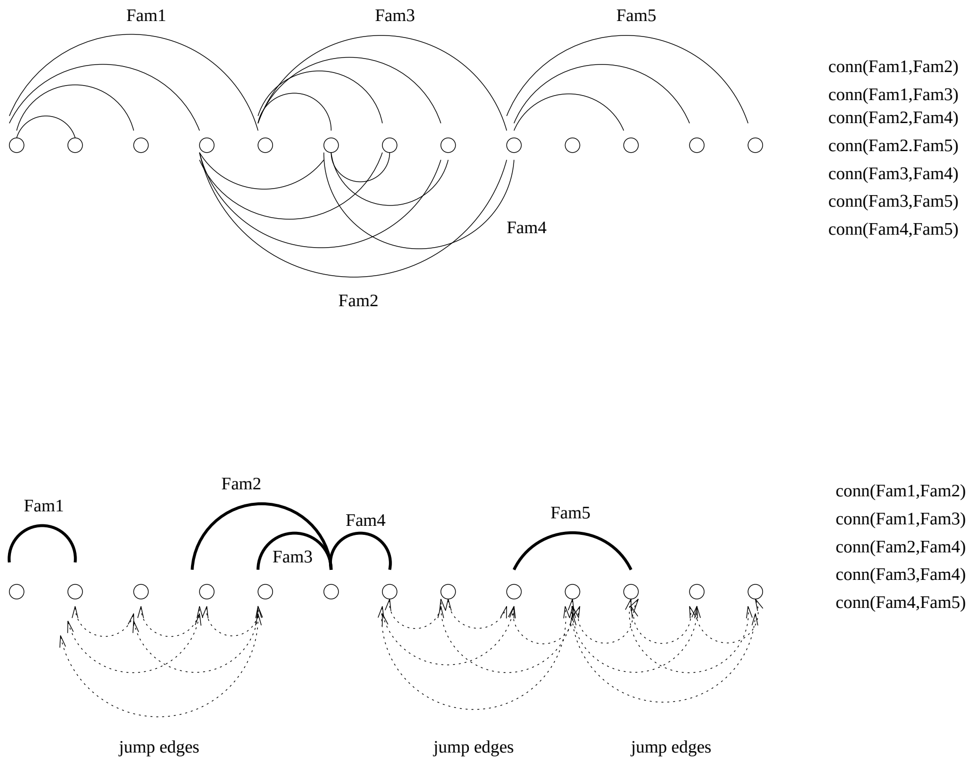

Abbildung 4.11: Das modifizierte Verfahren nach Chien et al. 1990.

Definition 4.9 *Ergänztes Verfahren nach Chien et al. 1990 ([16, 17]):*

1. Extraschritt, um Lattice ohne doppelte Worthypothesen zu erzeugen:

   Für eine gegebenen Menge von Worthypothesen, entferne für jede Familie F die Hypothesen f_2 bis f_n.

2. Verfahren nach Chien et al.:

   Für jedes Paar von Knoten v_i und v_j: Falls zwischen den Knoten keine volle terminale Kante existiert, erzeuge eine Sprungkante jump-edge(v_i, v_j)

Im ersten Schritt des Verfahrens wurde bewußt die jeweils erste Hypothese einer Familie ausgewählt, damit durch diesen Schritt nicht bereits eine LR-inkrementelle Vorgehensweise verhindert wird.

Die Abbildung 4.11 verdeutlicht die Wirkung des Verfahrens.

Das Beispiel aus Abbildung 4.11 ist absichtlich so gewählt, daß die Verbundenheitsrelationen zwischen Familien nicht erhalten bleiben. In den meisten Fällen, die wir untersuchten, blieben die Relationen erhalten. Die Heuristik, alle Knoten zu verbinden, wenn keine terminale Kante interveniert, garantiert dies jedoch nicht, wie unser Beispiel zeigt.

Das Verfahren konstruiert Verbundenheit nach einem heuristischen Kriterium. Dies ist nötig, solange für die erzeugte Lattice ohne Familien die ursprünglichen

Verbundenheitsbeziehungen nicht bekannt sind. In unserem Falle sind diese Relationen jedoch bekannt und fußen auf einem klaren Kriterium, wie in Definition 4.8 gegeben.

Dennoch ist die Vorgehensweise interessant, und das Einführen von Sprungkanten ist eine gute Möglichkeit, einmal erzeugte Kanten über mehrere Knoten zu teilen, sodaß Mehrfacharbeit vermieden werden kann.

Ein anderes, starreres Verfahren wird in Weber 1992 [95] verwendet. Die dort beschriebene Vorgehensweise ist vergleichsweise einfach, und gewährleistet kein optimales Ergebnis. Es ist aber mit einer LR-inkrementellen Vorgehensweise, zumindest für den einfachen Fall von BUI, verträglich. Zudem erhält es Reihefolgebeziehungen zwischen Familien, wie in Definition 4.8.

Definition 4.10 *Starres Time Mapping nach Weber 1992 ([95]):*

1. *Wähle einen Konstante I, sodaß keine Worthypothese kürzer sein kann als I Frames.*
2. *Erzeuge nur in jedem I-ten Zyklus einen neuen Knoten.*
3. *In jedem Parsezyklus: Falls für eine Worthypothese f_j bereits ein Vorgänger f_{j-1} im aktuellen Endknoten existiert, dann:*
   (a) *Falls der Score von f_j besser ist als der Score von f_{j-1}, dann ersetze den Score von f_{j-1} durch den Score von f_j. Korrigiere die Scores aller Folgekanten von f_j entsprechend. Füge f_j nicht in die Chart ein.*
   (b) *Sonst, füge f_j nicht in die Chart ein.*
4. *Sonst, füge f_j in die Chart ein.*

Das Verfahren von Weber 1992 faßt bis zu einer Schrittweite der Länge I alle Worthypothesen einer Familie zusammen. Damit kein Fehler hinsichtlich des Suchverhaltens entsteht, ist es nötig, immer die beste Bewertung zu propagieren. Daher muß gegebenenfalls eine Update-Operation über allen Folgekanten eines Familienmitglieds in einem Knoten ausgeführt werden. Dennoch wird die Strahlensuche durch das starre Time Mapping beeinflußt. Pruning von Paaren wird nur aufgrund von f_1 ausgeführt. Nur diejenigen Folgekanten, die dieses Pruning überleben, können nachher noch aktualisiert werden. Dies ist ein großer Nachteil des Verfahrens. Falls der Wert I zu groß gewählt wird, kann es passieren, daß sowohl der Anfangs- als auch der Endframe einer Worthypothese auf den gleichen Knoten abgebildet werden. Es entstünde dann eine Schlaufe, die wieder und wieder geparst werden kann. Im Experiment stellte sich ein Wert von 3 für I als sicher heraus und führte zu einer entspechenden Verringerung der Verarbeitungszeit. Ein solcher Wert ist jedoch stark von den verwendeten Wortmodellen und dem eingesetzten Decoder abhängig.

Während das heuristische Verfahren nicht sicher erscheint, in Bezug auf die Erhaltung der Reihenfolgebeziehungen, ist das starre Vefahren suboptimal im Hinblick auf eine Reduktion der Familien. Das starre Verfahren stört die Strahlensuche.[26] Das heuristische Verfahren verwendet Sprungkanten, um Knoten zu verbinden. Dies bedeutet, daß nicht zwischen zwei Familien unterschieden wird, die in ein und demselben Knoten enden. Falls Sprungkanten zu anderen Knoten existieren, so werden

[26] Im Experiment war der Störeffekt nur schwach.

immer beide Familien von anderen Knoten erreichbar. Dies ist unerwünscht und führt zu den Fehlern, wie im beschriebenen Beispiel aus Abbildung 4.11. Hingegen ist das starre Verfahren nur mit großem buchhalterischem Aufwand mit einer TDPI-Kopplung verträglich. Falls nur alle I Frames eine Prädiktion erzeugt und an den Decoder gesendet wird, tritt ein starker Glättungseffekt auf, der zu Erkennungsfehlern führen kann, da immer I Frames lang nicht mehr unterschieden werden kann, welche neuen Wortkopien vom Decoder gestartet werden sollten. Für BUITDV tritt hingegen mit beiden Verfahren kein Problem auf.

Im nächsten Abschnitt wird ein Verfahren dargestellt, das die Vorzüge beider skizzierter Ansätze vereint und die Nachteile vermeidet.

4.6.4 Inkrementelles Time-Mapping

Zunächst entwickeln wir in diesem Abschnitt die Idee der Sprungkanten weiter. Durch eine starke Modifikation des Ansatzes der Sprungkanten kann ein Verfahren formuliert werden, daß ein Time-Mapping optimal realisiert und links rechts inkrementell funktioniert. Grundvoraussetzungen des Chartparsing, wie etwa die Monotonie der Chart, bleiben dadurch erhalten. Es stellt sich jedoch heraus, daß die Verwaltung von Sprungkanten sehr aufwendig wird, wenn eine korrekte Metrik erhalten bleiben soll, sodaß das Suchverhalten noch dem des LR-ACP ohne Time Mapping entspricht. Die ursprüngliche Eleganz des LR-ACP geht dadurch verloren, indem z.B. die Fundamentale Regel stark verkompliziert wird.

Die endgültige Version des Algorithmus (4.14) verwendet schließlich keine Sprungkanten mehr, sondern simuliert durch ein Vererbungsverfahren den Effekt auf elegantere und effizientere Art. Indem auf Teile der Monotonieeigenschaften der Chart verzichtet wird, gelingt eine einfachere und dadurch überlegene Realisierung eines Time-Mapping.

4.6.4.1 Ein inkrementelles Time-Mapping mit Sprungkanten.

Beim heuristischen Verfahren der Sprungkanten war die Grundidee, für jede Familie nur ein Mitglied wirklich zu parsen. Die Ergebnisse dieses Parsing sollen jedoch von anderen Knoten zugreifbar sein.

Diese Knoten können jedoch nach den übrigen Familienmitgliedern ausgewählt werden und nicht nur durch eine globale Heuristik. Auf diese Weise läßt sich die eine Hälfte der Fehlerursache beim modifizierten Verfahren von Chien et al. 1990 beheben.

Eine weitere Fehlerquelle in Bezug auf die Reihenfolgebeziehungen ist die Tatsache, daß Sprungkanten generell Knoten verbinden, anstatt Familien. Indem Sprungkanten für Familien spezialisiert werden und nicht allgemein für alle Familien in einem Knoten gelten, wird dies korrigiert.

In Abbildung 4.12 wird klar, wie sich eine explizite Repräsentation zweier Familien in einer Chart verhält im Verhältnis zu einer Darstellung mit familienspezifischen Sprungkanten.

Eine Familie wird dabei ersetzt durch ihren ersten Vertreter und eine Sprungkante für jeden weiteren Vertreter.

Ein vorläufiger Algorithmus für einen Zyklus des LR-ACP mit Time Mapping kann dann wie in Definition 4.11 skizziert werden. Hierbei werden jeweils neue

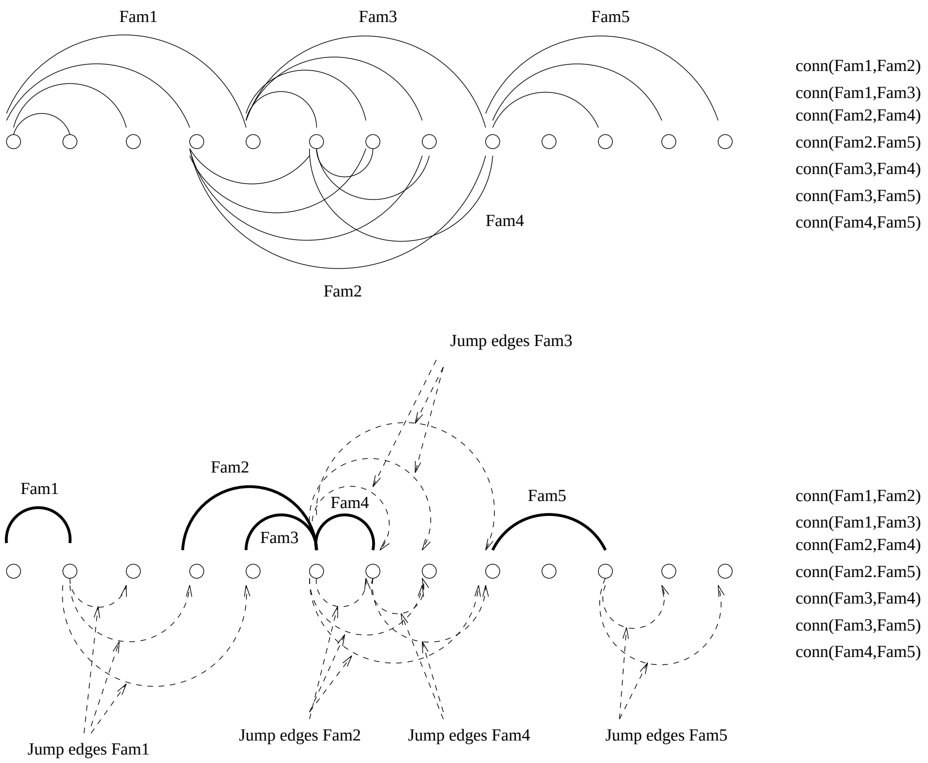

Abbildung 4.12: Explizite Repräsentation zweier Familien vs. Darstellung durch Prototypen und familienspezifische Sprungkanten.

Worthypothesen anders behandelt also solche, für die im Vorgängerknoten bereits ein Familienmitglied endet. Neue Hypothesen werden normal geparst, wobei jedoch die bereits existierenden Sprungkanten berücksichtigt werden. Für bereits bekannte Familien wird für einen neuen Vertreter einfach eine Sprungkante bis zum ersten Mitglied der Familie eingeführt.

Die Voraussetzungen für die Verwaltung der Sprungkanten sind folgende:

- Eine Funktion Families(v) liefert für einen Knoten v die Familiennamen der Worthypothesen zurück, die in v enden.
- Eine Funktion Successors(F) liefert diejenigen Kanten zurück, die durch die erste Hypothese einer Familie erzeugt wurden.
- Die Funktion Family(W) liefert für die Worthypothese W den Namen der Familie zurück, zu der W gehört. Familien sind eindeutig nach Anfangsknoten und Kategorien.
- Jump(v, F) ist eine Sprungkante für die Familie F, die vom Knoten v zum Endknoten der ersten Hypothese aus F geht.
- Prototyp(F) ist die erste Hypothese einer Familie F.

Definition 4.11 *Kontrollschleife für LR-ACP mit Sprungkanten:*

1. *Erzeuge Knoten v_0 und setze die Startkante(n) ein*[27]*.*
2. *Erzeuge Knoten v_1 und setze T auf 1.*
3. *Lies alle Worthypothesen mit Endezeitpunkt T.*
4. *Für alle aktuell eingelesenen Hypothesen W:*
   Falls $Family(W) \notin Families(v_{T-1})$:
   (a) *Füge W in die Chart ein und erzeuge einen neuen Familiennamen $[cat(W), begin(W)]$.*
   (b) *ACTIVE seien alle aktiven Kanten, die in $begin(W)$ enden, zuzüglich der aktiven Kanten A, für die folgendes gilt:*
   $\exists Jump(begin(W), F)$ und $A \in Successors(F)$.
   (c) *Stoße alle Tripel (A, W), $A \in ACTIVE$ auf die AGENDA.*
5. *Sonst: Füge eine Sprungkante $Jump(v_T, Family(W))$ ein.*
6. *Prune.*
7. *Parse.*
8. *Inkrementiere T, erzeuge Knoten v_T und gehe zu Schritt 3.*

[27] Dadurch wird eine SEEK-DOWN-Operation ausgelöst.

Der Algorithmus führt exakt für jede Familie einmal ein Parsing und Pruning aus. Für alle Folgeknoten verweist eine Sprungkante zurück auf den Prototypen f_1. Insofern verhält er sich korrekt bezüglich der Erhaltung der Reihenfolgebeziehungen.

Das erste Problem des Algorithmus 4.11 ist die gerechte Bewertung von Einträgen auf der AGENDA eines Zyklus für $a \in$ ACTIVE aus den beiden Teilschritten.

Der Schritt 4b – das Aufsammeln aller aktiven Vorgänger einer Worthypothese zu der Menge ACTIVE – zerfällt jetzt in zwei Teile. Das Aufsammeln der unmittelbaren Vorgängerkanten und das Aufsammeln derjenigen aktiven Kanten, die über Sprungkanten erreichbar sind.

Die Paare mit aktiven Kanten aus dem ersten Teilschritt sind gerecht bewertet. Die Paare mit aktiven Kanten aus dem zweiten Teilschritt jedoch nicht. Die folgende kurze Überlegung verdeutlicht dies.

Wir wollen annehmen, ein Paar der Agenda wurde im LR-ACP ohne Time Mapping durch eine aktive Kante A und die Worthypothese W gebildet. Die letzte Worthypothese, die von A überspannt wird, heißt f_i und ist i-tes Mitglied einer Familie F von Worthypothesen.

Der LR-ACP in Algorithmus 4.11 bildet entsprechend dem Paar (A,W) ein Paar (A',W), wobei A' und W durch eine Sprungkante von $v_{\mathrm{end}(A')}$ bis $v_{\mathrm{end}(A')+i}$ für die Familie F verbunden wird.

In die Bewertung von A geht also der akustische Score der Hypothese $f_i \in F$ ein. Jedoch geht in die Bewertung von A' der akustische Score der Hypothese $f_1 \in F$ ein.

Um das Bewertungsschema anzupassen, sodaß also A' im LR-ACP mit Time Mapping durch Sprungkanten den jeweils korrekten Score trägt, wie in den vergleichbaren Schritten des normalen LR-ACP, muß der akustische Anteil des Scores einer aktiven Kante eines Paares auf der Agenda durch die jeweilige Sprungkante modifiziert werden.

Definition 4.12 *Eine Sprungkante $Jump(v, Family(W))$, die für die Worthypothese W eingeführt wurde, erhält als Bewertung $logP(W|HMM)$, den akustischen Score von W.*

In die Agenda werden schließlich nicht mehr Paare sondern Tripel eingetragen. Die zusätzliche Stelle enthält eine Sprungkante, falls es eine gab, oder Nil. Schließlich wird für die Bewertung des Tripels, wie auch für die gegebenenfalls entstehende neue Kante, die aus dem Tripel hervorgeht eine neue OUTSIDE-Bewertung der Aktiven errechnet, indem der Score von A denormalisiert wird, die akustische Bewertung von Prototyp(W), durch die akustische Bewertung der Sprungkante ersetzt wird und der Score wieder normalisiert wird über der neuen Gesamtlänge von A und der Sprungkante. Auf diese Weise können auch diejenigen Einträge auf der Agenda gerecht bewertet werden, die durch Sprungkanten entstehen.

Ein weiteres Problem des Algorithmus 4.11 ist die Verträglichkeit mit der Strahlensuche. Im LR-ACP ohne Sprungkanten wird jedes Familienmitglied in einem anderen Parsezyklus bearbeitet. Es hat nicht nur eine andere Bewertung, auch das Maximum in jedem Zyklus ist verschieden. Somit werden in jedem Zyklus möglicherweise verschiedene Paare von aktiven Kanten und Familienmitgliedern beim Pruning verworfen.

In Algorithmus 4.11 kommt jedoch nur der Prototyp einer Familie überhaupt in einem Zyklus auf die Agenda. Alle Paare mit dem Prototypen, die beim Pruning verworfen werden, können auch in späteren Knoten nicht mehr geparst werden, auch wenn folgende Familienmitglieder noch in der akustischen Bewertung steigen.[28].

Ein zusätzlicher Mechanismus kann dieses Problem lösen. Wir fügen zu Schritt 5 einen weiteren Schritt hinzu, um die verworfenen Paare des letzten Knotens im neuen Knoten wieder in den neuen Pruningvorgang aufnehmen zu können. Vorausgesetzt wird das folgende:

- Die Funktion Nonparsed(F, v_i) gibt für eine Familie F und einen Knoten v_i diejenigen Paare der Agenda von Parsezyklus i zurück, die unterhalb des Beams lagen.

Definition 4.13 *Revision von Schritt 5:*

1. *Sonst: Füge eine Sprungkante Jump$(v_T, Family(W))$ ein.*
2. *Für alle Paare (A, X) in Nonparsed$(Family(W), v_{T-1})$, stoße ein entsprechendes Paar (A, W) auf die AGENDA.*

Der Algorithmus 4.11, entsprechend modifiziert, produziert nun für eine Familie Folgekanten in verschiedenen Knoten.[29] Dies ist in Abbildung 4.13 dargestellt.

Die Strahlensuche ist nunmehr korrekt. Es entsteht durch den Mechanismus in der revidierten Fassung von Schritt 5 jedoch ein neues Problem. Es werden zwar alle Nachfolger von Familienmitgliedern gebildet und nur genau einmal. Die Sprungkanten verweisen jedoch nur auf den Knoten des ersten Familienmitglieds. Die anderen Folgekanten, die sich jetzt auf andere Knoten verteilen, sind nicht mehr vom Ende jedes $f \in F$ erreichbar. Es ist also nötig, weitere Sprungkanten einzufügen, um wiederum diese Verbindungen herzustellen. Für jeden Knoten, in dem eine Worthypothese einer Familie endet, muß eine Sprungkante zu jedem Vorgängerknoten der Familie eingeführt werden.

So wird jedoch die Idee der Sprungkanten ad absurdum geführt. Es entsteht jetzt eine Inflation von Sprungkanten, was nicht nur bezüglich des Speicherbedarfs ungünstig ist.

Die Fundamentale Regel des ACP wird jetzt komplizierter, da die Sprungkanten immer mitverfolgt werden müssen. Beim Aufsammeln der neuen Einträge für die Agenda muß für jede Familie eine große Zahl von Sprungkanten zurückverfolgt werden. Schließlich muß jedesmal wenn ein solcher Zugriff über eine Sprungkante erfolgt, die neue korrekte Bewertung des Agendaeintrages errechnet werden. Katastrophal wirkt sich die große Menge der Sprungkanten ebenfalls auf den Redundanztest bei SEEK-DOWN Operationen aus. Alle Knoten die durch Sprungkanten erreichbar sind müssen dabei überprüft werden.

Obwohl also mithilfe von Sprungkanten eine Lösung für das Problem der Familien skizziert werden kann, ist jene unelegant und nicht effizient.

Die meisten der erwähnten Nachteile tauchen in der folgenden Variante eines LR-ACP mit Time Mapping durch Vererbung gar nicht erst auf.

[28] Das Maximum ist fast nie der Prototyp.

[29] Indem hier nur die Paare mit W in die neue Agenda eingetragen werden, wird doch ein kleiner Teil des Parsing eventuell doppelt ausgeführt. Es ist ein Nachteil dieses Verfahrens. Diesen Fehler zu vermeiden, wäre sehr aufwendig. In der Version des Time Mapping durch Vererbung in Abschnitt 4.14 taucht dieser Fehler nicht mehr auf.

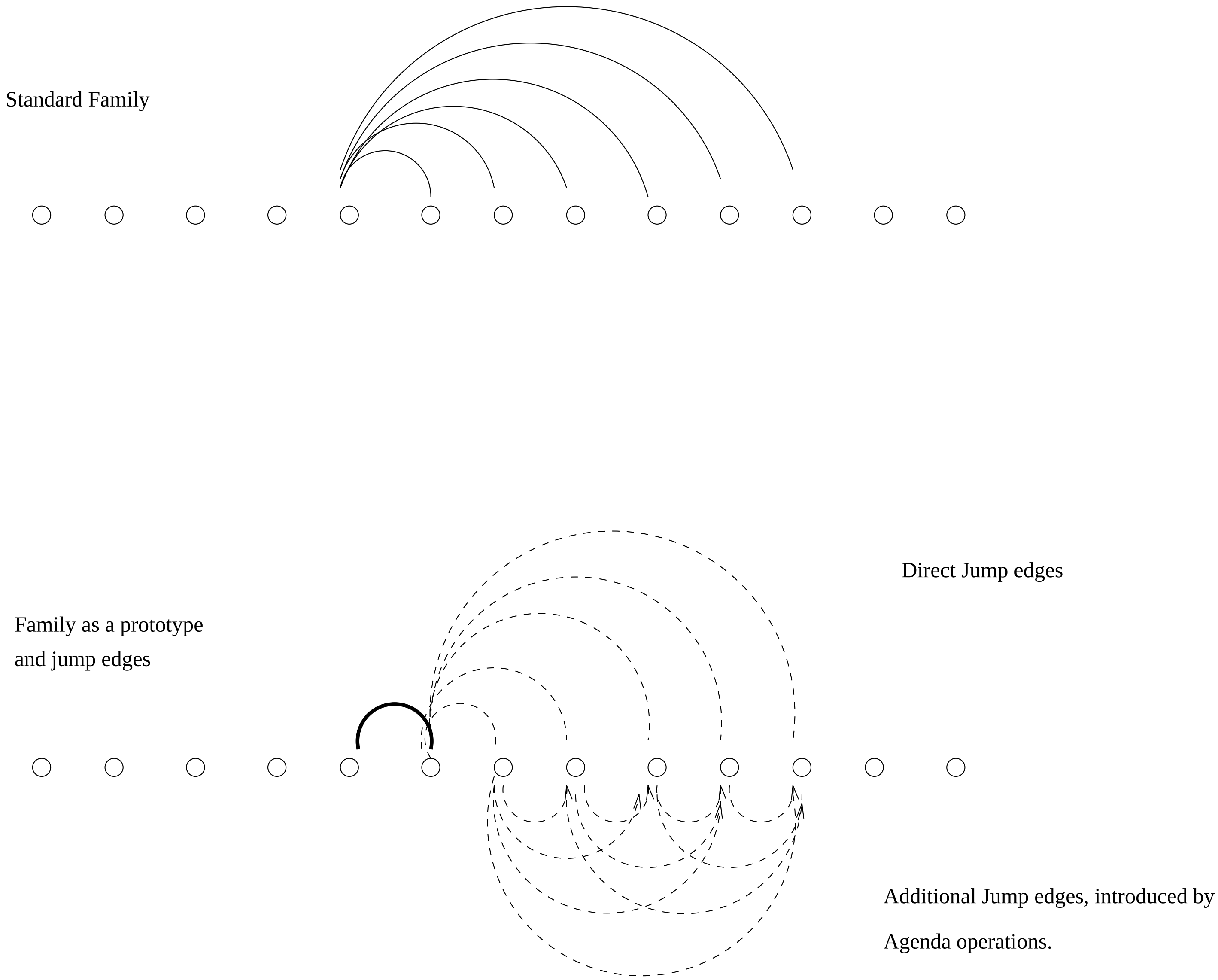

Abbildung 4.13: Eine Familie und die Folgekanten in jedem ihrer Knoten.

4.6.4.2 Inkrementelles Time Mapping durch Kantenvererbung.

Für eine elegantere Lösung des Problems werden zwei Grundannahmen des Chartparsing fallen gelassen:

- Bisher war die Chart eine monotone Datenstruktur. Im folgenden erlauben wir das Ersetzen von Kanten durch andere Kanten.
- Bisher war eine Kante immer genau einem Anfangsknoten und genau einem Endknoten zugeordnet. Im folgenden erlauben wir, daß Kanten eine Menge von Endknoten besitzen können, bzw. daß eine Menge von aufeinanderfolgenden Knoten sich eine Kante teilen kann.

Diese Veränderungen sind nötig, um den Effekt der Sprungkanten — eben das Teilen von Kanten durch Knoten — auch ohne Sprungkanten darstellen zu können.

Das folgende Verfahren stellt eine Familie von Worthypothesen nur noch als eine Kante dar. Kanten, die im Vorgängerknoten bereits erzeugt wurden, werden in einem Zyklus an den aktuellen Knoten *vererbt*. Danach gehört die Kante zu beiden Knoten. Aktueller Knoten einer Kante ist jedoch immer der neue Knoten[30]

Paare auf der Agenda, die im Vorgängerknoten verworfen wurden, werden für den nächsten Knoten nochmal auf die Agenda gelegt.

Das Maximum für das Pruning wird als Maximum der geerbten Kanten und dem Maximum der neuen Einträge auf der Agenda gebildet.

Ausgenommen von diesem *Vererbungsmechanismus* sind nur die leeren aktiven Kanten. Würden sie vererbt, so wären sie keine "Schlaufen" mehr, könnten also nicht mehr rekursiv angewendet werden. Außerdem haben die leeren Kanten die für die Suche wichtige Eigenschaft, daß ihre OUTSIDE-Bewertung eine obere Schranke für folgende Completer-Operationen darstellt. Sie erfahren eine Extrabehandlung, wobei wir eine neue SEEK-DOWN-Operation definieren müssen.

Ein Parsezyklus zerfällt grob in vier Schritte:

INHERIT Vererbe alle Worthypothesen und Nachfolger, für die bereits Vorgänger existieren. Ignoriere leere Kanten.

PARSE-NEW Behandle alle Worthypothesen, für die kein Vorgänger existiert, und berücksichtige dabei die beim Pruning verworfenen Konfigurationen mit bekannten Hypothesen des Vorgängerzyklus.

UPDATE-EMPTIES Vererbe die leeren aktiven Kanten, die es nicht schon gibt. Für leere aktive Kanten im letzten und im aktuellen Knoten, ersetze alle neuen, falls die alten besser bewertet sind.

Im Schritt INHERIT wird der Liste der Endknoten einer zu vererbenden Kante nur ein neuer Knoten hinzugefügt. Die akustische Bewertung einer solchen Kante wird solange aktualisiert, bis das Maximum der Familie erreicht ist. Die Kante wird außerdem in den neuen Knoten eingefügt, ohne daß sie aus dem Vorgängerknoten entfernt wird. Durch wiederholte Anwendung dieses Vererbungsschrittes entsteht also eine Reihe von Knoten, die alle ein und dieselbe Kante enthalten. Die Kante

[30] Wir stellen dies so dar, daß in der Implementation eine Kante eine Liste von Knoten enthält, wobei ein neuer aktueller Knoten immer als neuer Kopf der Liste "angeconst" wird.

selbst enthält eine Liste dieser Knoten, wobei der Kopf dieser Liste immer der aktuelle neue Knoten ist.

Im Schritt PARSE-NEW werden die neuen Worthypothesen wie im standard LR-ACP behandelt. Für jede Worthypothese wird mit allen aktiven Vorgängerkanten ein Paar auf die Agenda gelegt. Die Auswahl der aktiven Vorgängerkanten erfolgt dabei über die Anfangsknoten der Worthypothesen. Wenn also im standard LR-ACP ein solcher Knoten ein Familienmitglied und dessen Nachfolger enthielt, so enthält jetzt — durch den Vererbungsmechanismus — der entsprechende Knoten die eine prototypische Kante der Familie und deren prototypische Nachfolger.

Die Fundamentale Regel bleibt hier (fast) unverändert. Beim erfolgreichen Kombinieren zweier Kanten wird eine neue Kante in die Chart eingefügt. Die neue Kante erhält den Anfangszeitpunkt der beteiligten aktiven Kante und die Liste der Endezeitpunkte der beteiligten inaktiven Kante. Die neue Kante wird nur in den Anfangsknoten und den aktuellen Endknoten eingetragen.

Falls die neue Kante aufgrund eines Agendaeintrags entsteht, der im Vorgängerknoten beim Pruning verworfen worden war, so erhält die neue Kante die Liste der Endknoten der beteiligten Inaktiven[31], wird jedoch nur in den aktuellen Knoten eingetragen. Von diesem Knoten aus kann die Kante nunmehr in möglichen Folgezyklen weitervererbt werden. In den Vorgängerknoten wurde diese Kante ja verworfen, daher wird sie dort nicht eingetragen und kann von diesen Knoten aus auch nicht zugegriffen werden.

Falls die neue Kante aufgrund einer neuen Worthypothese zustande kam, entspricht der Mechanismus dem des standard LR-ACP, da die Liste der Endknoten nur ein Element enthalten kann, nämlich den aktuellen Endknoten.

Die leeren Kanten sind im Chartparser Gegenstand von Strukturteilung. Für die Vererbung muss ein Teil der "Packung" aufgefaltet werden, und zwar der Teil, der ersichtlich macht, durch welche Kanten eine leere aktive Kante hätte eingeführt werden können.

Eine leere aktive Kante wird immer durch die beste (erste) aktive Kante eingeführt, die eine entsprechende SEEK-DOWN-Operation auslöst. Spätere SEEK-DOWNs aufgrund anderer aktiver Kanten führen die selbe leere aktive Kante nicht noch einmal ein. Dies wird durch den Redundanztest verhindert.

Durch eine modifizierte SEEK-DOWN-Operation führen wir zwar weiterhin leere aktive Kanten nur einmal ein. Falls jedoch eine Kante — und damit ihre Nachfolger ebenso — bei einem Redundanztest erreicht wird, notieren wir dies in einem hierfür vorgesehenen Feld der leeren aktiven Kante. Das Feld *Families* enthält die Menge von Familiennamen, aufgrund derer sie eingeführt worden sein könnte.

Im Schritt UPDATE-EMPTIES wird schließlich auf dieses Feld Bezug genommen. Wenn der Schnitt der Menge zu vererbender Familien und das entsprechende Feld einer leeren aktiven Kante nicht leer sind, wird die leere Kante vererbt. Das entsprechende Feld wird bei der Vererbten mit der Schnittmenge initialisiert. Wirklich eingefügt in einen neuen Knoten wird eine Kopie der zu vererbende Kante.

Der Schritt UPDATE-EMPTIES muß zuletzt ausgeführt werden.

Wenn der Schritt PARSE-NEW ausgeführt ist, haben eventuell SEEK-DOWN-Operationen stattgefunden, d.h. daß für eine gegebene Regel bereits eine leere aktive Kante im aktuellen Knoten existiert. Für jede der zu vererbenden leeren Kanten

[31] Diese Liste ist ja bereits im Schritt INHERIT aktualisiert worden. Die beteiligte Inaktive muß ja in v_{T-1} existiert haben und vererbt worden sein, sonst gäbe es den Agendaeintrag nicht.

wird nun überprüft, ob nicht schon eine neue leere Kante im neuen Knoten erzeugt wurde. Falls dies zutrifft und die neue Kante besser bewertet ist als die alte, wird die alte Kante nicht vererbt. Falls die neue Kante schlechter bewertet ist als die alte, wird sie entfernt und durch eine Kopie der alten ersetzt.

Leere aktive Kanten werden also nicht im eigentlichen Sinne vererbt, sondern wirklich kopiert. Der Redundanztest, der für leere Kanten immer nötig ist, wird dabei ausgeführt, sodaß immer das Maximum überlebt. In diesem Schritt werden in Algorithmus 4.14 gegebenenfalls Kanten aus der Chart durch andere ersetzt – ein nichtmonotoner Schritt.

So ist gewährleistet, daß die leeren Kanten immer die obere Schranke einer OUTSIDE-Bewertung mitführen, also Nachfolger des besten parsbaren Pfades bis zum aktuellen Knoten darstellen.

Der folgende Algorithmus setzt zusätzlich Folgendes voraus:

- Für jeden Knoten K gibt Inactive-In(K) die Liste mit inaktiven Kanten zurück, die in K enden.
- Für jeden Knoten K gibt Active-In(K) die Liste mit aktiven Kanten zurück, die in K enden.
- Für jeden Knoten K gibt Empty-Active-In(K) die Liste mit leeren aktiven Kanten zurück, die in K enden.
- Für jeden Knoten K gibt Edges(K) die Liste aller in einem Knoten endenden Kanten zurück.
- Family(W) gibt den Namen der Familie von Worthypothesen zurück zu der W gehört.
- Ac-Score(F,V) gibt den acoustischen Score der Worthypothese von Familie F im Knoten V zurück.
- Für eine leere aktive Kante E enthält E.FAMILIES die Menge der Familiennamen, durch welche die Kante eingeführt wurde[32].
- E.RIGHT-V enthält die Liste der Endknoten der Kante E.
- E.RULENUMBER enthält die eindeutige Nummer der Regel der Unifikationsgrammatik der Kante E.
- E.acoustic-o-operations enthält die Anzahl Frames, die von der Worthypothesenfolge überdeckt wird, die zu akustischen Anteils des Outside-Scores von E führten. Entsprechend definiert seien E.acoustic-i-operations, E.n-gram-o-operations, usw.
- E.acoustic-o-value enthält die Summe akustischen Scores der Worthypothesen, die zu akustischen Anteils des Outside-Scores von E führten. Entsprechen definiert seien E.acoustic-i-value, E.n-gram-o-value, usw.

Definition 4.14 *LR-ACP mit Time-Mapping durch Vererbung*

[32] Bzw. eingeführt worden wäre, wenn kein Redundanztest stattgefunden hätte.

1. *Erzeuge Knoten v_0 und setze die Startkante(n) ein*[33].
2. *Erzeuge Knoten v_1 und Setze T auf 1.*
3. *Lies alle Worthypothesen W mit Endezeitpunkt T.*
4. *NEU sei $\emptyset$. ALT sei $\emptyset$.*
5. *Für alle W:*

   Falls eine Vorgängerhypothese in Knoten v_{T-1} existiert, füge W in ALT ein.

   Sonst, füge W in NEU ein.
6. *INHERIT:*
   (a) *Für jedes W in ALT*
      i. *Für jede Kante E in Inactive-In(v_{T-1}):*
         Falls $E \in$ Successors(Family(W)) dann BEGIN
         E.RIGHT-V := cons(v_T,E.RIGHT-V),
         UPDATE-ACOUSTIC-SCORE(E,W),
         Inactive-In(v_T) := cons(E,Inactive-In(v_T))
         END.
      ii. *Für jede Kante E in Active-In(v_{T-1}):*
         Falls $E \in$ Successors(Family(W)) und nicht leer ist, dann BEGIN
         E.RIGHT-V := cons(v_T,E.RIGHT-V),
         UPDATE-ACOUSTIC-SCORE(E,W),
         Active-In(v_T) := cons(E,Active-In(v_T)).
         END
      iii. *Stoße alle Paare in Nonparsed(Family(W),v_{T-1}) auf die AGENDA*
7. *PARSE-NEW*
   (a) *Für jedes W in NEU:*
      i. *Füge W in die Chart ein.*
      ii. *Stoße jedes Paar (A, W) auf die AGENDA, sodaß $A \in$ Active-In($v_{from(W)}$)*
   (b) *MAXVALUE := MAX (MAX (Score(E), $E \in$ Edges(v_T)), Score(Top(AGENDA)))*
   (c) *Parse die AGENDA bis zur Beamschwelle.*
   (d) *SAVE-AGENDA*
8. *UPDATE-EMPTIES*
   (a) *EMPTIES sei $\emptyset$*
   (b) *Für jede Kante E in Inactive-In(v_{T-1}):*
      Falls E leer ist, und falls gilt: E.Families $\cap \{F \mid F = Family(W), W \in ALT\} \neq \emptyset$
      EMPTIES := EMPTIES $\cup$ $\{E\}$

[33] Dadurch wird eine SEEK-DOWN-Operation ausgelöst.

(c) *Für jede Kante E aus EMPTIES*
   i. *E' := copy(E)*
   ii. *E'.Families := E.Families* $\cap\{F|F = Family(W), W \in ALT\}$
   iii. *UPDATE-EMPTYSCORE(E')*
   iv. *Falls eine Kante A aus Empty-Active-in(v_T) existiert sodaß E.RULENUMBER = A.RULENUMBER:*
      A. *Falls Score(E') > Score(A):*
         Ersetze A durch E'
         E'.Families:= E'.Families $\cup$ *A.Families*
      B. *Sonst: A.Families := E'.Families* $\cup$ *A.Families*
   v. *Sonst: Füge E' in V_T ein.*

Die wichtigen Operationen UPDATE-ACOUSTIC-SCORE und UPDATE-EMPTIESCORE sind unten angegeben. Während das Vererben von nicht leeren Kanten direkt plausibel ist, da – wie in Abschnitt 4.6.2 gezeigt wurde – zwei terminale Kanten aus einer Familie exakt dieselben Completeroperationen haben können, ist die Handhabung der leeren Kanten kompliziert.

UPDATE-ACOUSTIC-SCORE sorgt für die korrekte Behandlung der nicht leeren vererbten Kanten. Die Korrektheit dieses Schrittes ist direkt ersichtlich. Zwei Komponenten gehen in die Bewertung von Kanten ein: Die Bi-Gramm-Bewertung bleibt unberührt, da die vererbte Kante die gleiche Wortkette überspannt, wie die ursprüngliche. Die akustischen Worthypothesen, die von den vererbten nicht leeren Kanten überspannt werden, sind ebenfalls die gleichen – bis auf die letzte Worthypothese, diese wird ersetzt durch die neue Worthypothese der gleichen Familie.

Definition 4.15 *UPDATE-ACOUSTIC-SCORE(E,W)*

1. *Increment E.acoustic-o-operations*
2. *E.acoustic-o-value :=*
   Normalize(
   Denormalize(E.acoustic-o-value)
   + Ac-Score(Family(W),V_T)
   - Ac-Score(Family(W),V_{T-1}))
3. *Increment E.acoustic-i-operations*
4. *E.acoustic-o-value :=*
   Normalize(
   Denormalize(E.acoustic-i-value)
   + Ac-Score(Family(W),V_T)
   - Ac-Score(Family(W),V_{T-1}))

Die Behandlung der leeren Kanten ist komplexer. Zunächst muß entschieden werden, welche Kanten vererbt werden müssen. Wie oben bereits kurz erwähnt, geben wir zu diesem Zweck jeder leeren Kante ein Feld E.Families.[34]. Leere Kanten habe zwei wesentliche Eigenschafte, die unsere Vorgehensweise nahelegen.

[34] Das Füllen des Feldes E.Families bei jeder Seek-Down-Operation ist einfach und für eine Programmierübung geeignet. Da wir Seek-Down-Operationen vorkompiliert verwenden, ist die Aufgabe noch einfacher.

- Eine leere Kante wird aufgrund des Best-First Zugriffs auf die Agenda von der besten aktiven Kante eingeführt, die sie einführen kann.
- Eine leere Kante im aktiven Chartparser repräsentiert auch die entsprechenden Teilschritte der top-down Parses anderer, schlechter bewerteter Pfade durch Strukturteilung.

Wenn die Worthypothese, die zu den bestbewerteten Kanten führte, nicht vererbt wurde, jedoch andere Worthypothesen, deren schlechter bewertete Nachfolger zu den selben leeren Kanten geführt hätten, so muß die entspechende leere Kante vererbt werden. Sie wäre nämlich beim original LR-ACP im nächsten Knoten aufgrund einer der schlechteren nicht leeren Kanten eingeführt worden, die in der Time Mapping Version vererbt werden.

Wenn eine leere Kante kein Nachfolger irgendeiner zu vererbenden Kante hätte sein können, dann wird sie nicht vererbt. In diesem Fall ist der Schnitt von zu vererbenden Worthypothesen und möglichen Einführern der betreffenden leeren Kante leer.

Um die erste obige Eigenschaft von vererbten leeren Kanten zu erhalten, muß der Score bei der Vererbung neu berechnet werden. Dies wird in der Operation erreicht UPDATE-EMPTIESCORE erreicht, indem über den vererbten Worthypothesen beüglich des kombinierten Scores, der mit ihnen gebildet werden kann, maximiert wird.

Definition 4.16 *UPDATE-EMPTIESCORE(E)*

1. *E.intro ist* $w_1..w_n$.
2. *MAXWORD :=*
   $Argmax\ (W \mid \gamma\ \frac{LogP(w_1 \ldots w_{n-1} W | N-Gram)}{n}$
   + Normalize(Denormalize(E.acoustic-o-val)
   + Ac-Score(Family(W),V_{T-1}*)*
   - Ac-Score(Family(w_n*),*V_T*)))*
3. *E.intro :=* $w_1..w_{n-1}$ *MAXWORD*
4. *E.n-gram-value :=*
   Normalize(
   Denormalize(E.n-gram-value)
   - LogP($w_1..w_n|$ *N-Gram)*
   + LogP($w_1..w_{n-1}$ *MAXWORD | N-Gram)))*
5. *E.acoustic-o-value :=*
   Normalize(Denormalize(E.acoustic-o-val)
   + Ac-Score(Family(MAXWORD),V_T*)*
   - Ac-Score(Family(w_n*),*V_{T-1}*)))*

In der Tabelle 4.14 ist ein Effizienzvergleich für BUI mit Time Mapping durch Vererbung und ohne Time Mapping angegeben. Dabei wurden im Parser und im Decoder Beams mittlerer Breite eingesetzt.

	Time in Sek.	Edges
Mit TM	68.8	6613
Ohne TM	334.4	13854

Abbildung 4.14: BUI mit und ohne Time Mapping

Es wurden dabei nur Sätze gezählt, die erkannt wurden. Die Versionen mit und ohne Time Mapping erkannten in allen Experimenten exakt die selben Sätze. Dies klingt trivial, ist jedoch nicht selbstverständlich. Der LR-ACP mit Time Mapping durch Veerbung verhält sich bezüglich des Scoring verschieden zur Standardversion.[35]

Für eine Familie von Worthypothesen ist das typische Verhalten in Abbildung 4.10 dargestellt worden[36]: Die Bewertung steigt von f_1 an, erreicht ein Maximum und fällt wieder ab. Bei der Version mit Time Mapping fällt eine Familie nicht ab. Alle Mitglieder nach dem Maximum erhalten den Wert des Maximums. Dies kann im Falle von Wortkopien dazu führen, daß die global maximierende Erkennung der Wortgrenzen zu anderen Resultaten führt. Indem Familien nur durch eine prototypische Kante dargestellt werden, darf nach dem Maximum die Bewertung nicht mehr verändert werden. Dies entspricht einer LR-inkrementellen Variante der Viterbi-Strategie, wie sie bereits bei Ney 1991 [65] nicht-inkrementell vorgenommen wurde. Ney läßt für eine Familie von Worthypothesen generell nur das Maximum überleben. In seiner Formulierung eines CYK-Parsers für Mengen von Worthypothesen wird generell auf diese Weise bottom up maximiert. In unserem Ansatz wird, inkrementell von links nach rechts gesehen, die gleiche Technik verwendet. Der Unterschied ist, daß immer nur der linke Kontext bis T bekannt ist, in einem Parsezyklus zur Zeit T. Wir maximieren nicht global, sondern nur nach dem Teil des globalen Kontexts, der zum Zeitpunkt T bekannt ist. A posteriori betrachtet landen jedoch alle Kanten beim Maximum. Die Bereiche, in denen unterhalb des globalen Maximums einer Familie gearbeitet wird, sind lokal.

Der Faktor 5, der sich in den Experimenten ergab, ist unterhalb der durch das Verhältnis von Hypothesen zu Familien gegebenen Schätzung für einen zu erwartenden Speed Up. In unserer Implementation des Vererbungsmechanismus wurde keine letzte Optimierung vorgenommen. Daß sich das Verhältnis der Kanten nicht entsprechend dem Verhältnis der Verarbeitungszeit verhält, kommt durch die leeren aktiven Kanten. Ihre Zahl (etwa ein fünftel aller Kanten ohne Time Mapping) bleibt durch das Time Mapping unverändert. Hingegen ist der Prädiktorschritt vorherberechnet. Die leeren Kanten verbrauchen also wenig Rechenzeit.

Die Completer-Operation – also die Suche nach links – ist die eigentlich teure Operation beim Parsing. Diese wird durch das Time Mapping betroffen. Denn diese Operation führt alle nicht leeren Kanten ein, außer den direkt durch Worthypothesen und Lexikonzugriff erzeugten passiven Kanten. Sämtliche Kanten, die durch den Completer-Schritt erzeugt werden, werden also vererbt.

[35] Bis auf diesen Unterschied stellt er jedoch eine präzise Nachbildung dar.

[36] vergl. Abschnitt 4.6.1.

4.6.5 Modifikation des inkrementellen Time Mapping für BUITDV

Für den Einsatz mit BUITDV muß das Time Mapping etwas anders organisiert werden, ohne daß im wesentlichen andere Mechanismen angewendet werden.

Ein Parsezyklus zerfällt bei BUITDV grundsätzlich in zwei Teile. Der Decoder schickt zuerst die zu verifizierenden neuen Worthypothesen und dann erst – wenn er die Bestätigungen erhalten hat – die bekannten Worthypothesen desselben Frames, für die er bereits eine Verifikationsnachricht im vorangegangenen Zyklus erhalten hatte. Dies liegt daran, daß für das Pruning innerhalb des Decoders immer nur verifizierte Wortendehypothesen verglichen werden sollen.

Der Schritt des Auseinandersortierens von neuen und bekannten Hypothesen entfällt also für den Parser. Dafür wird der Mechanismus komplizierter, mit dem die alte beim Pruning verworfene Agenda gerettet wird und mit aktualisierter Bewertung nochmal betrachtet wird. Ein Zyklus des LR-ACP ist schließlich wie in Definition 4.17 beschrieben. Die Details sind hier nicht spezifiziert, da sie exakt denen des Algorithmus 4.14 in Abschnitt 4.6.4.2 entsprechen.

Definition 4.17 *Ein Parsezyklus des LR-ACP für BUITDV mit Time Mapping durch Vererbung*

1. *Neue Hypothesen:*
   (a) *Empfange alle zu verifizierenden Hypothesen.*
   (b) *Baue die Agenda auf und Parse. OLD-MAX sei das lokale Maximum.*
   (c) *Verifiziere PRUNE-AGENDA.*
2. *Bekannte Hypothesen:*
   (a) *Erbe alle entsprechenden nichtleeren Kanten aus dem Vorgängerknoten. INH-MAX sei deren Maximum.*
   (b) *Nimm PRUNE-AGENDA des Vorgängerknotens und lege alle entsprechenden aktualisierten Kanten auf die AGENDA. Nimm als Maximum das Maximum von OLD-MAX und INH-MAX.*
   (c) *Parse. Rette PRUNE-AGENDA.*
   (d) *Aktualisiere die leeren Kanten.*

Für BUITDV ist durch das Time Mapping zwar eine Effizienzsteigerung zu bemerken, nicht jedoch in der gleichen Weise wie bei BUI. Der Grund ist, daß das Time Mapping genau bei einer Schwäche des BUI greift, die bei BUITDV nicht so stark ausgeprägt ist.

Bereits im Abschnitt 4.5 wurde bemerkt, daß die Qualität der Worthypothesen für BUITDV relativ schlecht ist.[37] Der Parser übernimmt hier ja die Aufgabe mit, für den Decoder die Wortübergangsstrafen zu errechnen. Dies bedeutet, daß der Parser für jeden Wortübergang des Decoders auch viele Hypothesen erhält, die später unter den Beam des Decoders fallen, wenn der Decoder die Wortübergangsstrafe des Parsers auf das Wortmodell aufaddiert.

[37] Zur Erinnerung: Qualität war Verhältnis von bottom up Hypothesen zu Kanten.

	Time in Sek.	Edges
Mit TM	51.4	4662
Ohne TM	115.8	9765

Abbildung 4.15: BUITDV mit und ohne Time Mapping

	Time in Sek.	Edges
Mit TM	172.6	5004
Ohne TM	282.7	7519

Abbildung 4.16: Standard TDPI mit und ohne Time Mapping

Wie ist aber dann das gute Abschneiden von BUITDV bei den Experimenten in Abschnitt 4.5 überhaupt zu erklären?

Bei BUITDV wirkt relativ wenig top down Restriktion vom Parser auf den Decoder. Die Maxima der konkurrierenden Zustände waren durchweg sogar höher als bei BUITDV. Die Hauptrestriktion, die den Speed Up von BUI zu BUITDV bewirkt, liegt darin, daß der Parser seinen eigenen Suchraum nach rechts beschränkt. Indem er eine bottom up Hypothese verifiziert und die entsprechende Wortkopie aufgrund einer Verifikation mit schlechtem Score unter den Beam des Decoders rutscht, taucht sie nicht noch einmal als Hypothese auf.

Ohne Time Mapping muß eine solche Hypothese bei BUI wieder und wieder vom Parser in Betracht gezogen werden. Während bei engen Beams der Effekt schwächer ist und sogar zu einer schlechteren Erkennungsrate von BUITDV führt, ist die Erkennungsrate bei weitem Beam gleich, bei BUITDV muß der Parser jedoch viel weniger suchen.

Die gleiche Ersparnis – nämlich daß kritische Worthypothesen nur einmal geparst werden – ergibt sich durch das inkrementelle Time Mapping jedoch sowieso.

Dies erklärt, wieso das Time Mapping bei BUI wesentlich mehr Gewinn erzielt als bei BUITDV. Dennoch ist ein Gewinn deutlich messbar. Die Ergebnisse über 5 Sätzen im Vergleich zu BUITDV ohne Time Mapping sind in Tabelle 4.15 gegeben.

4.6.6 Ein inkrementelles Time Mapping für die Kalkulation von Prädiktionen.

Der LR-ACP mit inkrementellem Time Mapping durch Vererbung, wie in Algorithmus 4.14 gegeben, ist voll mit TDPI verträglich. In Tabelle 4.16 ist die Wirkung des Time Mapping bei TDPI zu sehen. Sie ist wie zu erwarten messbar, jedoch noch geringer als bei BUITDV, denn der Anteil der Kalkulation der top down Prädiktion am Rechenzeitbedarf von TDPI ist groß, während das Time Mapping nur das eigentliche Parsing effizienter macht.

Für die Kalkulation der Prädiktion kann jedoch die gleiche Technik angewendet werden, wie für das eigenliche Parsing. Wenn eine Familie von Worthypothesen mehrere Endpunkte belegt und aufgrund dieser Familie eine oder mehrere Mitglieder einer Prädiktion entstehen, so wird auch hier die gleiche Kalkulation mehrfach ausgeführt.

Zunächst können wir die folgenden prototypischen Fälle lokalisieren, in denen die Vorgehensweise offensichtlich ist:

- Von einem Knoten i zum Folgeknoten j ändert sich nichts an den ankommenden Worthypothesen. In diesem Fall kann die Prädiktion von j durch die Prädiktion von i ersetzt werden.

- Von einem Knoten i zum Folgeknoten j ändern sich alle Hypothesen. In diesem Fall muß die Prädiktion neu berechnet werden.

Zu diskutieren ist also der dritte Fall, wenn einige Hypothesen aus i in Knoten j enthalten sind, jedoch auch neue hinzukommen und/oder einige fehlen.

Aufgrund des Time Mapping-Algorithmus kann immer zwischen neuen Kanten und solchen, die vom Vorgängerknoten geerbt wurden, unterschieden werden.

Indem wir also für die geerbten Kanten die jeweilige Information speichern, welche Nachfolgewörter durch sie vorgeschlagen wurden, müssen zumindest diese nicht neu berechnet werden.

Um Doppelarbeit zu vermeiden, ist es nötig, solche durch das N-Gram vorgeschlagene Verlängerungen von neuen Kanten nicht mehr zu überprüfen, die bereits durch die geerbten Kanten ableitbar sind.

Zusätzlich werden für den Test-Schritt (Schritt 9 von Algorithmus 4.5) nur noch die neuen nicht leeren aktiven Kanten (und die leeren Kanten) verwendet.

Wie gewohnt klären wir zunächst die Erweiterungen und Hilfsfunktionen für die Spezifikation des Algorithmus

- Für jede nichtleere aktive Kante A wird ein neues Feld A.PREDS eigeführt. In dieses Feld werden während der Berechnung einer Prädiktion die top down Hypothesen eingetragen, die aufgrund von A im Testschritt lizensiert werden. Das Feld enthält initialisiert die leere Liste.

- Für jede Kante A wird ein neues Feld A.INH eingeführt. Dieses Feld steht auf T, wenn die Kante schon einmal vererbt wurde, sonst auf nil.

- P_i ist die für den Knoten i in Zyklus i berechnete Prädiktion.

- Die Funktion NOCHANGE(i,j) liefert T, falls die Worthypothesen von Knoten i die gleichen waren wie von Knoten j.

Definition 4.18 *Die Berechnung von Prädiktionen P mit Time Mapping für Knoten v_i:*

1. *Falls NOCHANGE(v_{i-1}, v_i), set $P_i := P_{i-1}$.*

2. *Sonst, $P_i := \emptyset$*

   (a) *INH-ACTIVES seien alle aktiven nichtleeren Kanten A, mit A.INH = T.*
   NEW-ACTIVES seien alle aktiven nichtleeren Kanten A, mit A.INH = nil.
   EMPTIES seien alle aktiven leeren Kanten.
   INH-PREDS := $\emptyset$.

	Time in Sek.	Edges
TDPI mit TM und TM-Prädiktion	58.2	4226
TDPI mit TM und standard Prädiktion	172.6	5004

Abbildung 4.17: Time Mapping vs. standard Kalkulation der Prädiktion bei TDPI

(b) *Für alle A in INH-ACTIVES:*
INH-PREDS := CONS(A.PREDS, INH-PREDS).

(c) *Bilde P' wie in Algorithmus 4.5, Schritt 2 bis Schritt 5. NEA wird dabei ersetzt durch NEW-ACTIVES.*

(d) *Entferne aus P' alle Kandidaten, die in INH-PREDS enthalten sind.*

(e) *Führe Schritt 6 bis 9c von Algorithmus 4.5 aus zur Bildung von P. ACTIVES wird dabei ersetzt durch NEW-ACTIVES ∪ EMPTIES.*

(f) *P := P ∪ INH-PREDS.*

Falls kein Unterschied bezüglich der Familien in zwei Knoten besteht, erbt der Algorithmus sofort die Prädiktion des Vorgängerknotens.

Falls alle Hypothesen neu sind, ergo keine Kantenvererbung stattgefunden hat, dann ist INH-PREDS leer. In diesem Falle entspricht der Algorithmus 4.18 dem Verfahren des Algorithmus 4.5

Das Aufsammeln der Prädiktionen, die durch die vererbten Kanten erzeugt wurden, in Schritt 2b ermöglicht die direkte Vererbung zur nächsten Prädiktion. Die Arbeitsersparnis kommt, indem durch das Filtern in Schritt 2d die Vererbten nicht nochmal zur Überprüfung herangezogen werden. Zusätzlich werden auch nur noch die neuen Kanten für den Überprüfungsschritt 2e verwendet

Die Tabelle 4.17 zeigt die Wirkung des Time Mapping für die Berechnung einer Prädiktion.

Kapitel 5

Integration der Methoden

Dieses Kapitel ist der Verwendung der Ableitungswahrscheinlichkeiten der Unifikationsgrammatik als statistisches Modell gewidmet.

Die Vorgehensweise wird zunächst motiviert. Danach werden einige Ansätze in dieser Richtung dargestellt. In der Vergangenheit gibt es nur wenige Ansätze, die sich mit der Kopplung dieser Paradigmen befassen. Wir versuchen eine systematische Vorgehensweise, indem wir eine Definition einer probabilistischen Unifikationsgrammatik vorschlagen, die die Details der Kopplung von Ableitungen und Modell offen läßt. Diese Details werden im anschließenden Abschnitt eingehender betrachtet, wobei unserer Meinung nach die verwendeten Mittel für die Kopplung vom verwendeten Typmechanismus der Grammatik abhängen sollten.

Für typisierte Unifikationsgrammatiken führen wir eine spezielle Vorgehensweise ein, um die Ableitungen und Typoperationen probabilistisch zu modellieren.

Schließlich wird eine Variante von Fujisakis[1] Trainingsverfahrens für PCFG angegeben, mit der sich diese Art von probabilistischer typisierter Unifikationsgrammatik (PTUG) trainieren läßt.

Am Ende der Arbeit werden einige Experimente mit dem LR-ACP und der PTUG diskutiert.

5.1 Probabilistische Unifikationsgrammatik

5.1.1 Motivation für eine probabilistische Variante von Unifikationsgrammatiken.

Für die Verwendung einer probabilistischen Unifikationsgrammatik für den LR-ACP gibt es drei Hauptargumente.

- Systeme, wie die Kopplungen im Abschnitt 4 als ganzes gesehen, modellieren eine Abbildung f:S $\rightarrow$ MS, wobei s $\in$ S ein Schallsignal ist und ms $\in$ MS eine Merkmalsstruktur. Wenn wir nur das N-Gramm-Modell und die akustischen Bewertungen zur Suche verwenden, erhalten wir immer noch eine Menge möglicher ms als Ergebnis. Gewünscht ist aber noch eine Disambiguierung

[1] Vergl. Einführung.

zwischen verschiedenen Ableitungen, die durch die Unifikationsgrammatik für eine gefundene Worthypothesenfolge erzeugbar ist.

- Durch ein zusätzliches probabilistisches Modell kann eine bessere Erkennungsleistung erreicht werden.

- Durch ein Modell der grammatischen Operationen wird die Strahlensuche des LR-ACP nach links verbessert. Dadurch sollte eine gleiche Erkennungsleistung trotz weniger Unifikationsoperationen erreicht werden.

Auf die genannten drei Aspekte wird bei den durchgeführten Experimenten in Abschnitt 5.3.2 wieder Bezug genommen.

Zusätzlich zu den rein technischen Argumenten paßt die Vorgehensweise natürlich in unser Paradigma der Kopplung von Parser und Decoder. Bisher konnte nur die Parsbarkeit als Filter zur Verfügung gestellt werden, der nicht auch bereits vom Decoder mit N-Gramm-Modell alleine verfügbar gewesen wäre. In den beiden top down Szenarien des vergangenen Kapitels kann mit einer PUG dem Decoder eine zusätzliche Beschränkung zur Verfügung gestellt werden, die ohne LR-inkrementelle Kopplung nicht realisierbar wäre. Der Übergang von probabilistischen CFGs zu probabilistischen Unifikationsgrammatiken bietet einige Vorteile:

Kontextfreie Grammatiken sind bekanntlich für die Modellierung von Semantik, Dialog oder Übersetzung nicht geeignet.[2]

Darüber hinaus gibt es jedoch auch aus statistischer Sicht Vorzüge von probabilistischen Unifikationsgrammatiken.

- Bei Unifikationsgrammatiken werden viele natürliche Klassen, wie z.B. nur bezüglich Kongruenz verschiedene Phrasen durch die selben Regeln modelliert. Wir finden also bereits eine starke Generalisierung vor. Bekannte und offensichtliche Generalisierungen können hier also von Hand in die Grammatik geschrieben werden.

  Weniger offensichtliche oder domänenabhängige Regularitäten können schließlich über die statistische Komponente hinzugefügt werden.

- Kontextfreie Grammatiken haben typischerweise sehr viele Regeln, um ein entsprechendes Fragement abzudecken. Beim Grammatiktraining generell ist zu beobachten, daß die Menge der Regeln immer der Stichprobe "davonläuft". Um mehr Daten abzudecken, müssen wieder mehr Regeln zugefügt werden, dadurch entstehen mehr zu trainierende Parameter und die Stichprobe muß vergrößert werden, usw..

  Unifikationsgrammatiken weisen um Größenordungen kleinere Mengen von Regelinstanzen auf. Durch linguistisch motivierte Zusammenfassung reduzieren sich die zu trainierenden Parameter für das statistische Modell. Dies entspricht dem Effekt einer Klassenbildung, wie etwa für N-Gramme verwendet.

[2] Da hierauf von vielen Autoren seit Jahren hingewiesen wurde, wollen wir diesen Punkt nicht weiter vertiefen.

5.1.2 Bisherige Ansätze für enge Kopplungen von Statistik und Unifikationsgrammatik.

Im folgenden werden die bisherigen Ansätze kurz dargestellt, bei denen eine enge Kopplung von Unifikationsgrammatik und probabilistischem Modell realisiert wurde.

Zunächst sind hier die Arbeiten vom MIT im Kontext des Verarbeitungmodells TINA zu erwähnen. Mit TINA bestehen bereits Erfahrungen für die Verarbeitung gesprochener Sprache.

Einige Arbeiten, wie etwa Hemphill & Picone 1989 [41] versuchten bereits, probabilistische Unifikationsgrammatiken einzuführen. Die dortige Vorgehensweise ist jedoch sehr vereinfachend.

Darüber hinaus sind mit PEARL und PICKY probabilistische chartbasierte Ansätze, mit denen merkmalsbasierte Grammatikformalismen verarbeitet werden können, vorgeschlagen worden. Diese Arbeiten sind zwar nicht an gesprochener Sprache erprobt, stellen aber die derzeit geradlinigste Vorgehensweise dar. In Magermann 1994[3] [59] werden verschiedene Möglichkeiten diskutiert, statistische Modelle über grammatische Ableitungsfolgen kontextsensitiv zu gestalten. Dies ist für probabilistische Unifikationsgrammatiken von höchster Relevanz.

Eine wichtige Arbeit in diesem Bereich ist Briscoe & Carrol 1993, die einen probabilistischen GLRP für die Verarbeitung einer Unifikationsgrammatik verwenden. Briscoe & Carrol 1993 greifen wichtige Probleme auf, wie etwa die Konstruktion von kontextfreien Approximationen von Unifikationsgrammatiken, ein Problem der Abbildung unendlich vieler Ausprägungen auf endlich viele Klassen.

Der in unserer Arbeit verfolgte Ansatz geht etwas anders vor als die oben erwähnten. Indem wir — dem Stand der Forschung entsprechend — typisierte Unifikationsgrammatiken verwenden, ergibt sich ein etwas anderes Problem der Handhabung eines kontextfreien Rückgrats einer Grammatik.

Wir behandeln typisierte Unifikationsgrammatiken, indem wir eine Abbildung des Typsystems auf ein Bi-Gramm-Modell vornehmen. In bezug auf die Debatte in Magermann 1994 zu probabilistischen Abhängigkeiten bei Ableitungen halten wir diese Vorgehensweise für einen guten Kompromiß für das Parsing von Lattices.

Hierfür werden schließlich einige Eigenschaften diskutiert, wie etwa die Verträglichkeit mit diversen Eigenschaften von Typsystemen. Schließlich werden noch angepaßte Varianten des Fujisaki-Trainingsverfahrens für unserer Zwecke angegeben.

5.1.2.1 Einfache Ansätze mit annotierten PCFGs

Der TINA Parser, beschrieben in Seneff 1989 [81] ist der erste "echte" probabilistische Language-Parser, der für gesprochene Sprache entwickelt wurde, insofern er mehr als nur eine einfache kontextfeie Grammatik verwendet.

In TINA werden annotierte kontextfreie Regeln verwendet, die zum Parsen in eine Netzwerkstuktur gepackt werden. Das statistische Modell von TINA ist ein standard PCFG-Modell: Die Grammatik ist sauber in kontextfreie Regeln mit eigenen Variablen und in "Filter" – den Feature-Anteil der Grammatik — geteilt.

[3] Die beste Arbeit, die ich seit Jahren las.

Für TINA wird ein einfaches Modell verwendet, daß Regelwahrscheinlichkeiten direkt aus der Häufigkeit und der Normalisierung über alle Regeln der gleichen linken Variablen ermittelt.

Die "Filter" beschränken dabei die möglichen Parses und beeinflussen auf diese Weise die Statistik der Regelanwendungen.

Die von uns in Abschnitt 5.2.2 vorgeschlagene Handhabung von Typen resultiert in einem Modell wie TINA, wenn der Typverband flach ist, also alle Typen (wie CFG-Variable) unverträglich miteinander sind.

Eine ähnlich Vorgehensweise betreiben Hemphill & Picone 1989 [41]. Sie präsentieren bereits eine (leider widersprüchliche) Definition für PUG, die wir hier wiedergeben, da an ihr das Dilemma eines zu einfachen Vorgehens sichtbar wird.

Definition 5.1 *Definition einer PUG nach Hemphill & Picone 1989 [41], (Seite 723): 'Definition: A stochastic unification grammar is a fourtuple* $G_s = (V_N, V_T, P_s, S)$, *where* V_N *and* V_T *are finite sets of nonterminals and terminals,* $S \subset V_N$ *is the set of start symbols, and* P_S *is a finite set of stochastic productions each of which is of the form* $A, p \rightarrow \alpha$, *where* $A \in V_N$, p *is the probability of applying the rule and* $\alpha \in (V_N \cup V_T)^*$. *Let the set of probabilities of all* k *stochastic productions in* P_s *with* A *on the left be* $\{p_i | A, p_i \rightarrow \alpha, i = 1, ..., k\}$. *Then* $0 < p_i \leq 1$ *and* $\sum_{i=1}^{k} p_i = 1$. *The nonterminals and terminals are feature value pairs."*

"Definition: A feature set is set of feature-value pairs of the form f:V, where f is a constant (0-ary function symbol) and V is either a constant, a variable, or a feature pair........"

Hemphill & Picone 1989 übernehmen direkt eine Definition für eine probablistische kontextfreie Grammatik. Diese Definition ist streng daran gebunden, daß nur endlich viele Variablen existieren.

An anderer Stelle der Definition wird dann gefordert, daß alle Variablen der Grammatik Merkmalsstrukturen sind, und es wird schließlich eine Definition von Merkmalsstrukturen angeboten, die unendlich viele Ausprägungen von MS erlaubt.

5.1.2.2 GPLRP

In der Arbeit von Briscoe & Carroll 1993 [13] wird das Problem der Nutzbarmachung von PCFG-Techniken eingehender betrachtet. Für die DCG[4], die sie in ihrem Ansatz verwenden, und die unendlich viele mögliche Kategorienausprägungen erlaubt, schlagen sie eine Methode vor, um ein kontextfreies Rückgrat zu extrahieren, sodaß neue Kategorien erzeugt werden, die mit den Merkmalsstrukturen korrespondieren. Diese Vorgehensweise wird unten weiter diskutiert.[5]

Die Autoren verwenden einen Generalisierten Probabilistischen LR-Parser (GPLRP) um eine DCG zu verarbeiten. Dabei wird nicht, wie z.B. in Arbeiten von Wrigley & Wright 1991 [99] eine bereits trainierte Grammatik[6] in eine LALR(1)-Tabelle umgewandelt, sondern die Übergänge und die Aktionen pro Übergang des Automaten werden mit einem einfachen Modell durch Abzählen trainiert.

[4] Definite Clause Grammar, siehe Einführung.

[5] In der Einführung, Abschnitt 2.3.1 ist ja bereits ein solches Verfahren erläutert.

[6] Wrigley & Wright 1991 verwenden nur eine PCFG und keine Unifikationsgrammatik. Für eine kontextfreies Rückrat einer Grammatik wäre die Erstellung der Tabelle natürlich identisch.

Es resultiert eine Variante eines Markovmodells, das mit einer Viterbisuche durchlaufen werden kann.

Für einen Vorteil ihres Modells gegenüber Ansätzen mit direktem Training der Regelinstanzen halten die Autoren, daß gegenüber direkt an Regeln geknüpfte probabilistische Modelle, wie bei Fujisaki et al. 1991 [33] oder Jelinek 1992 [44], ein größerer Kontext durch das Modell in die Entscheidungen eingeht. Dies ist nicht generell korrekt. So wie Briscoe & Carroll 1993 die statistische Variante eines GLRP von Wright 1990 [98] um Transitionswahrscheinlichkeiten erweitern, kann auch mehr Kontextsensitivität in direktes Grammatiktraining aufgenommen werden. Charniak & Carroll 1994[7] [15] zeigen dies z.B. für PCFGs mit kontextsensitivem Modell und präsentiert eine entsprechende Version des Inside-Outside-Algorithmus.

Grundsätzlich ist die Methode, direkt den Tomita-Automaten zu trainieren, interessant, bringt jedoch keine unmittelbaren Vorteile gegenüber regelbasierten Modellen, die ebenfalls kontextsensitiv organisiert sein können.

5.1.2.3 PEARL und SPATTER

In Magermann 1994 [59] werden Möglichkeiten diskutiert, durch die Verwendung von mehr Kontextinformation zu einem probabilistischen Grammatikmodell zu gelangen, das ohne explizite Verwendung von Merkmalsstrukturen die gleiche desambiguierende Information implizit im probabilistischen Modell kodiert.

Als Eckpunkte der Möglichkeiten für probabilistische Grammatikmodelle erscheinen dabei die PCFG, die keinen Kontext modellieren und das *History Based Grammar Model* (HBGM), bei dem die Wahrscheinlichkeit einer Regelanwendung immer von allen vorhergehenden Schritten einer Linksableitung abhängt.

Bei HBGM, wird durch einen Beweisbaum-basierten Klassifikator entschieden, welche Klasse von globalem Kontext in einer durch eine Grammatik gegebene Ableitung lokal vorliegt.

Schließlich wird aus den Ergebnissen mit HBGM in Magermann 1994 das SPATTER Parsing entwickelt. Hierbei entscheidet das Modell nicht mehr zwischen einigen durch eine Grammatik gegebene Menge von Ableitungsbäumen. Bei SPATTER bewertet das Modell alle möglichen Bäume, die über einer Kette von Wörtern kombinatorisch möglich sind und erzeugt so eine Ableitung ohne explizite Grammatik.

Dadurch wird ein völlig neues Paradigma eingeführt. Die Erzeugung einer Ableitung ist nicht mehr *generativ*, sondern bei SPATTER *selektiv*. Bäume werden nicht mehr abgeleitet, sondern nur noch ausgewählt.

Die grundlegende Erkenntnis aus Magermann 1994 ist, daß durch ein geeignetes kontextsensitives Modell die grammatischen Operationen überflüssig gemacht werden können.

Wichtiger für unsere Arbeit jedoch ist die Sichtweise auf grammatische Ableitungen. Alle Modelle, die Magermann 1994 diskutiert, identifizieren diskrete Ereignisse über möglichen Ableitungen, um an diese ein probabilistisches Modell zu knüpfen.

Beim Pearl-Modell[8] wird z.B. eine Vorgehensweise für aus PATR-artigen Unifikationsgrammatiken erzeugte CFG-Skelette vorgeschlagen, das stark an die Vorschläge zur Bildung eine kontextfreien Skeletts von Briscoe & Caroll 1993 erinnert.

[7] Anm: Es ist nicht der selbe "Carroll". John C. arbeitet am GPLRP, während Glenn C. die kontextsensitive PCFG betreibt.

[8] In Magermann 1994 diskutiert, aber bereits in Magermann & Markus 1991 [58] veröffentlicht.

Verarbeitet wird die Grammatik jedoch in einem Chartparser.

Das SPATTER-Modell ist zwar interessant, läßt sich jedoch für unserer Zwecke der Integration nicht naheliegend verwenden. Unser Ansatz bemüht sich ja um eine enge Verschränkung von Suche und Analyse. Die Unifikationsgrammatik kann hier nicht wirklich ersetzt werden, da sie in einem echten System die Basis für weitere Verarbeitungsschritte, wie etwa eine detaillierte semantische Verarbeitung bildet. Die Vorgehensweise bei Pearl käme jedoch durchaus in Frage. Jedoch wird auch bei Pearl das eigentliche Parsing nur noch mit dem kontextfreien Gerüst ausgeführt. Unsere Vorgehenseweise ist derjenigen aus Pearl durchaus ähnlich. Die Operationen im Parser werden jedoch mit der ursprünglichen Grammatik ausgeführt und nicht mit der kontextfreien Approximation.

5.1.3 N-Gramm-Modelle der Ableitungen von UGs

Die meisten probabilistischen Modelle, die PCFGs zugrundeliegen, sind N-Gramm-Modelle über Ableitungsfolgen. In der Einführung[9] wurden die PCFGs über die Ableitungsrelation definiert. Durch die Ersetzungsrelation ergibt sich für jeden Schritt einer Ableitung ein Paar von linker und rechter Seite einer Regel. Diese kann als Bi-Gramm behandelt werden.

Die wesentliche Beobachtung für die Erweiterung von Ableitungs-N-Grammen auf Unifikationsgrammatiken ist die folgende:

- Für die Anwendung von Ableitungs-N-Grammen als Basis für ein probabilistisches Modell muß ein Grammatikmodell gegeben sein, das in diskreten Schritten Ableitungen bildet. Im Verlauf solcher Ableitungen dürfen nur endlich viele Klassen diskreter Zustände auftreten.

Wenn unendlich viele Klassen diskreter Zustände möglich sind, kann trivialerweise nicht mehr garantiert werden, daß sich für ein N-Gramm über solchen Ableitungen die Summe aller abhängigen Beobachtungen zu 1 aufaddiert.

Dies ist der Grund, warum die Definition einer PUG von Hemphil & Picone 1989 unsinnig ist. Die Autoren definieren ihr probabilistisches Modell direkt über MS, ohne auf das Klassenproblem einzugehen.[10]

Für die PCFG sind die Klassen des N-Gramms den nonterminalen Symbolen der Grammatik gleichgesetzt. Dies entspricht in Analogie einem N-Gramm-Modell für Wortketten, das keine Klassen kennt, sondern bei dem jede Wortform eine eigene Klasse bildet.

Der Schlüssel für die direkte Definition einer probabilistischen Unifikationsgrammatik ist also deren Ableitungsrelation und die Klassenbildung z.B. über Merkmalsstrukturen.

Wie bei Magermann 1994 müssen die diskreten Ereignisse jedoch nicht unbedingt Merkmalsstrukturen sein. Im HBGM werden u.a. Regelobjekte als Ereignis in einer Ableitung angesehen. Ihm ging es jedoch vor allem darum, Unifikationsgrammatiken zu ersetzen. Unsere Vorgehensweise versucht, Unifikationsgrammatiken effizienter verarbeitbar zu machen und zusätzliche Desambiguierungsmechanismen einzubeziehen.

[9] Vergleiche Abschnitt 2.1.2.

[10] Sicher haben die Autoren von [41] eine solche Klassenzuordnung bei Experimenten vorgenommen, jedoch in ihrer Darstellung evtl. als selbstverständlich vorausgesetzt.

Die Definition von diskreten Ereignissen, mit denen sich eine Ableitung beschreiben läßt, und ein statistisches Modell über solchen Ereignissen sind zwei verschiedene Dinge und müssen getrennt voneinander behandelt werden.

Über spezielle Vorgehensweisen läßt sich für probabilistische Unifikationsgrammatiken generalisieren:

Definition 5.2 *Eine probabilistische Unifikationsgrammatik ist ein 4-Tupel: (L,R,C,PM), wobei gilt:*

L: ist das Lexikon der Unifikationsgrammatik. (w,MS) $\in$ L sind Paare von Wortformen und Merkmalsstrukturen.

R: sind die Regelinstanzen der Unifikationsgrammatik.

C: ist eine Klassenzuordnung von während einer Ableitung beobachtbaren Objekten zu endlich vielen Klassennamen.

GM: ist ein probabilistisches Modell über Folgen von Klassen.

Im folgenden Abschnitt diskutieren wir verschiedene Möglichkeiten, bei durch Unifikationsgrammatiken erzeugten Ableitungen Klassen zu finden und – damit korrespondierend – die Frage nach dem geeigneten probabilistischen Modell.

5.2 Klassenannahmen und passende probabilistische Modelle

Im LR-ACP in dieser Arbeit wird eine typisierte Unifikationsgrammatik verwendet, die entweder das Typgerüst losgelöst von den Merkmalsstrukturen verwenden kann oder mit Typ –"Aproppriateness", wie in Carpenter 1991 [14], arbeitet.

Unifikationsgrammatiken unterscheiden sich stark voneinander. Für die verschiedenen Varianten sind nicht die gleichen Klassenannahmen über Ableitungsereignisse praktikabel. Wir unterscheiden in diesem Abschnitt drei Typen von Unifikationsgrammatiken.

Zu Grammatiken ohne Typsystem gibt es bereits Ansätze, die teilweise oben dargestellt wurden. Wir diskutieren hier noch einmal die möglichen Varianten.

Eingehender beschäftigen wir uns mit der Verwendung des Typsystems im darauffolgenden Abschnitt. Wir präsentieren einen Ansatz, bei dem das statistische Modell der Ableitungen vor allem auf den Operationen über Typen arbeitet, wobei Typvererbung und "Aproppriatness" möglich bleiben. Dieses Modell geht schließlich in Abschnitt 5.3.1 in den LR-ACP ein. In Abschnitt 5.3.2 werden Experimente präsentiert.

Anschließend diskutieren wir noch die Verträglichkeit des Ansatzes mit den mächtigsten üblichen Typsystemen – solchen mit Typinferenz.

5.2.1 Typfreie Unifikationsgrammatiken

Bei Unifikationsgrammatiken ohne Typen sind Variable der Grammatik reine MS. Zuerst betrachten wir die Klassenbildung für solche Merkmalsstrukturen und die Vorgehensweisen, die durch diese Abbildung PCFG-Techniken nutzbar machen.

5.2.1.1 Klassen von Merkmalsstrukturen durch Shiebers *Restriction*.

Bei typfreien Unifikationsgrammatiken gibt es für die Klassenbildung bereits die offensichtliche Vorgehensweise aus Briscoe & Carrol 1993, die auch schon 1991 für das Pearl-Modell von Magermann & Marcus verwendet wurde. Im Wesentlichen ist diese Methode mit der Technik der *Restriction* in Shieber 1985 [86] bereits gegeben worden und nichts neues.

Der einzige zusätzliche Verarbeitungsschritt ist die Umbenennung der *Restriction*-MS um aus dieser endlich großen Menge von MS (die Restrictions bezüglich ausgewählter Merkmale) Variablen einer kontextfreien Grammatik zu erzeugen. Im Wesentlichen korrespondiert diese Vorgehensweise auch mit dem informell in der Einführung angegebenen Verfahren zur Erstellung eines GLRP für eine Unifikationsgrammatik.[11]

Definition 5.3 *Abbildung von MS auf endliche viele Klassen durch Shiebers* Restriction

1. *Wähle eine Menge von Merkmalen aus, die immer atomare Werte haben und bilde alle Merkmalsstrukturen nur mit diesen Merkmalen und allen ihren möglichen Werten.*[12] *Vergib eindeutige Namen* $Name_1 ... Name_n$ *an all diese möglichen Instanzen von* Restrictions*.*

2. *Für jede Regel aus R klassifiziere alle MS von R danach, ob sie mit* Restriction $Name_i$ *unifiziert werden können.*

3. *Für jede Regel aus R für die MS in mehrere Klassen fallen: Erzeuge soviele Kopien von R, daß alle Namen eindeutig werden.*

4. *Für jede Regelkopie von Regeln aus R unifiziere* MS_j *mit Namen* $Name_k$ *der Regelkopie mit der* Restriction $Name_k$*.*

5. *Für das Lexikon der Grammatik*[13] *gehe man wie folgt vor: Die linken Seiten der Lexikonproduktionen werden wie alle anderen MS behandelt. Alle Strings für Wortketten bilden eine eigene Klasse.*

Ein solches Verfahren kann selbstverständlich in diversen Varianten formuliert werden. Der Zweck ist erfüllt, wenn alle Regelobjekte – und das Lexikon — so in Klassen aufgeteilt werden, daß die Mitglieder zweier verschiedener Klassen sich nicht gegenseitig subsumieren und auch nicht miteinander unifiziert werden können. Das evtl. Vervielfältigen von Regeln in Schritt 3 dient dazu, dies zu gewährleisten. Explizite (Disjunktionen) oder implizite (Unterspezifikation) Nichtdeterminismen müssen hierzu für einen Teil der Merkmale entfernt werden — der Teil der Merkmale für die *Restrictions*. Dies erlaubt schließlich, deterministisch zu entscheiden, in welche Klasse eine Merkmalsstruktur fällt.

Die probabilistischen Modelle, die hierdurch verwendet werden können, sind N-Gramme[14] über Klassen von MS.

[11] Vergl. Abschnitt 2.3.1.

[12] Hier müssen wirklich nur alle Werte erzeugt werden. Es darf keine Unterspezifikation auftauchen, sodaß etwa eine *Restriction* eine andere subsumieren kann.

[13] Die Lexikoneinträge betrachten wir hier als unäre Produktionen.

[14] Natürlich können hier auch andere Klassifikatoren, wie z.B. Magermanns Entscheidungsbäume [59] eingesetzt werden.

Systematisch betrachtet erlaubt diese Bildung der Klassen von Merkmalsstrukturen die folgenden Vorgehensweisen:

a) **Ersetzen der UG durch die erzeugte PCFG:** Man erzeugt durch die ursprüngliche Unifikationsgrammatik eine Baumbank über einem Korpus. Danach ersetzt man die Unifikationsgrammatik durch eine kontextfreie Grammatik, deren Variablen die Klassennamen der *Restrictions* sind. Für jede oben erzeugte Regelkopie enthält die CFG eine entspechende Regel. Man trainiert schließlich die CFG zu einer PCFG anhand der durch die Unifikationsgrammatik gegebenen Baumbank. Durch das überwachte Lernen gelangt die *Restriction* der ursprünglichen UG in die probabilistische Komponente der PCFG.

   Dies ist die Vorgehensweise von Pearl.

b) **Parsing der PCFG als Skelett für die UG:** Man geht vor wie oben, hält jedoch für jede Regel der PCFG einen Index auf die ursprüngliche Regel der UG.

   Es bleiben zwei Möglichkeiten:

   - Man zieht nach jeder Regelanwendung einer Regel der PCFG die entsprechende Regelanwendung der UG nach. Falls eine Regel der UG scheitert, setzt man die Wahrscheinlichkeit der entsprechenden PCFG-Operation auf 0.
   - Man parst mit der PCFG[15] in einem angemessenen Format[16]. Wenn ein Parse gefunden ist, bildet man ihn mit der Unifikationsgrammatik nach. Wenn der UG-Parse scheitert, erzeugt/nimmt[17] man den nächsten Parse mit der PCFG usw.

     Dies wäre die Vorgehensweise, die Briscoe & Carrol 1993 entspricht.

c) **Direktes Trainieren und Parsing mit den UG-Regelkopien:** Da wir für jede Regelkopie und jede ihrer MS wissen, zu welcher Klasse sie gehört, können wir die Kopien auch direkt trainieren. Das Training kann überwacht oder unüberwacht ausgeführt werden. Die Metrik kann wie für PCFG gestaltet werden. Alle linken Seiten, die zur selben Klasse gehören, müssen zu 1 aufaddierbar sein.

   Da bereits für jede Regelkopie die Klassen, zu denen die Mutter- und Töchter-MS gehören, bekannt sind, muß dies auch beim Parsing nicht überprüft werden. Mit einer bestimmten Regelkopie bleibt dies konstant. Insofern sind beim Parsing die Klassen immer direkt zugreifbar und Regelwahrscheinlichkeiten können direkt ausgerechnet werden.

d) **Direktes Training und Parsing der UG:** Training und Parsing können auch direkt mit den Regeln der original UG durchgeführt werden. Das probabilistische Modell bleibt in diesem Falle nur implizit an den Regelkopien verankert.

[15] Bestensuche

[16] Wir abstrahieren hier für einen Moment von speziellen Parsingalgorithmen.

[17] Hier können alle kontextfreien Parses erzeugt werden, oder bei Bedarf der nächste. Das hängt vom Parsingverfahren ab. Bei ACP geht es nach Bedarf. Bei CYK oder GLRP ist es günstiger, den gesamten gepackten Wald zuerst zu erzeugen. Bei Agenda-getriebenem GLRP, wie etwa derjenige von Staab 1994 [89], kann wie beim ACP vorgegangen werden.

Sowohl beim Training als auch beim Parsing muß für jede Regelanwendung entschieden werden, zu welcher Regelkopie-Klasse sie innerhalb eines Parses instanziiert. Dies kann erfolgen, indem jeweils Subsumptionstest mit den jeweiligen *Restrictions* durchgeführt werden, die für mit einer Regel assoziierten Regelkopien in Frage kommen.

Block (mündliche Kommunikation)[18] macht einen Vorschlag in dieser Richtung für die a posteriori Disambiguierung von durch CYK-Parsing enstehende gepackte Wälder von Unifikationsgrammatiken.

Es wird hier deutlich, daß in Definition 5.2 die Klassenbildung zu Recht vom verwendeten probabilistischen Modell abgetrennt wird.

Der Pearl-Ansatz, der ja nur die resultierende PCFG verwendet, kann eigentlich nicht als PUG aufgefaßt werden. Wir werden ihn nicht weiter diskutieren.

Beim Parsing mit dem Skelett und der a posteriori Unifikation der korrespondierenden Regeln der UG — wie in b) — hingegen handelt es sich beim PCFG-Skelett nur um eine effiziente Darstellung der Klassenvertreter, an denen die Modellwahrscheinlichkeiten schnell ausgerechnet werden können.

Wir haben in a)-d) absichtlich auf die Spezifikation von speziellen Parsingverfahren verzichtet. Der GPLRP von Brisoe & Carrol 1993 für Unifikationsgrammatiken ist eben nur eine mögliche Ausprägung der Vorgehensweise b). Mit einem Chartparser ist die gleiche Vorgehensweise vorstellbar. Entspechend verhält es sich nicht mit allen genannten Varianten. Für eine GLPR-orientierte Verarbeitung kommen nur a) und b) in Frage, da eine LR-Tabelle statische Variablen, die sich zur Laufzeit nicht verändern, voraussetzt.

Variante c) und d) sind an Earley- oder CYK-basierte Verfahren gebunden, die ohne ein CFG-Skelett direkt mit Unifikationsgrammatiken umgehen können.

Alle vier Varianten lassen diverse probabilistische Modelle zu, mit denen Beobachtungssequenzen erzeugt bzw. bewertet werden können. Entscheidend ist hier nur die jeweilige Zuordnung grammatischer Operationen und Objekte zu beobachteten Klassen.

Die Varianten c) und d) scheinen sehr ähnlich. Sie haben jedoch sehr verschiedene Eigenschaften, die für den Typ der verwendeten Grammatik entscheidend sind, aber auch für das Parsingverfahren.

Das Verfahren c) ist besser geeignet für Unifikationsgrammatiken, deren Regeln sehr konkret und zahlreich sind, während das Verfahren d) für solche Grammtiken sehr aufwendige Tests zur Laufzeit erfordert.

Für stark lexikalisierte Grammatiken, die wenige, sehr abstrakte Regelschemata aufweisen — also HPSG-ähnliche Grammatiken — ist das Verfahren d) überlegen. Ein Ausprägen der Regelkopien würde hier die ansonsten identischen Regeln vervielfachen. Im Vergleich zu der Vervielfachung erscheint der Aufwand für die Subsumptionstests zum Zwecke der Klassenbestimmung gering – im Besonderen, da ja nur wenige Regeln überprüft werden müssen.

Es stört im Übrigen nicht, daß für Grammatiken mit Relationen die Subsumption unentscheidbar ist.[19] Die Tests sind immer nur lokal für die Werte einiger Merk-

[18] Vortrag auf dem Workshop "Prozedurale Aspekte der Sprachverarbeitung" der KI 94, Saarbrücken, September 1994.

[19] Vergleiche Einführung.

male der *Restriction* und können durch eine nicht-destruktive Unifikation ersetzt werden.[20]

Ein Problem mit Verfahren d) besteht darin, daß möglicherweise die Merkmale, die durch den Test mit der jeweiligen *Restriction* abgefragt werden, zu diesem Zeitpunkt noch nicht deterministisch sind. Dies kann auch für einen gesamten Parse gelten, bei dem so ein Merkmal Disjunktionen oder Unterspezifikationen enthält, die nie mehr durch eine spätere Unifikation aufgelöst werden.

Dieses Problem kann gelöst werden, wenn Wahrscheinlichkeiten erst a posteriori berechnet werden, für bereits bestehende Parses. In diesem Fall kann eine Instanz einer Regel, die mit mehreren Regelkopien des statistischen Modells unifizierbar ist, die Summe der jeweiligen Modellübergänge, oder aber das Maximum erhalten.[21]

Für eine LR-inkrementelle Verarbeitung scheidet ein Verfahren wie d) jedoch bereits aus diesem Grund aus. Auch bei einer VITERBI-Suche, also dem Mitnehmen nur des Maximums, ist durch die nicht-Lokalität der Unifikationsoperationen mit Koreferenzen ein solches Verfahren nicht mehr sinnvoll.[22]

5.2.1.2 Klassen durch Regelobjekte

Eine sehr viel einfachere Methode, zu diskreten Beobachtungen während der durch eine UG erzeugte Ableitung zu gelangen, ist durch die Regelobjekte der UG selbst gegeben.

Für eine gegebene Linksableitung

$$S \Rightarrow \alpha \Rightarrow .....\beta \Rightarrow W \tag{5.1}$$

ist immer eindeutig identifizierbar, welche Regel in einem Schritt angewendet wurde, unabhängig von der konkreten Ausprägung. Die Ableitung kann also geschrieben werden als Folge von Regeln, wobei wir hier Lexikonproduktion zu den Regeln zählen:

$$R_{i,0,0}, R_{j,i,Dtr_m(i)}, R_{k,l,Dtr_n(l)} \ldots\ldots\ldots \quad \begin{array}{l} \text{,wobei } l = i \text{ oder } l = j, \\ 1 \leq m, n \leq \max(\text{lenght}(R)) \end{array} \tag{5.2}$$

Die Länge von R bezeichnet die Anzahl von Töchtern einer Regel R. Die Kodierung oben ist wie folgt zu verstehen:

Index 1 bezeichnet die Nummer einer Regel.

Index 2 bezeichnet die Nummer der Regel, deren Tochter durch die aktuelle Regel expandiert wurde.

Index 3 bezeichnet die fortlaufende Nummer dieser Tochter auf der rechten Seite der durch Index 2 bezeichneten Regel.

[20] Subsumptionstest oder nichtdestruktive Unifikation dienen hier beide dem gewünschten Zweck. Eigentlich ist dies nur eine Nachbildung eines Tests auf echte Identität.

[21] Abhängig davon, ob wir ein INSIDE-OUTSIDE- oder VITERBI- Suchkriterium anwenden wollen.

[22] Es ist mit großem buchhalterischen Aufwand trotzdem möglich: Für jede Kante wird vermerkt, welche Vorgängerkanten bezüglich der Klasse nicht eindeutig waren. Bei jeder Unifikation wird dann getestet, ob sich daran was geändert hat. Danach werden die Bewertungen entsprechend aufgefrischt. Diese Lösung ist einfach nicht schön.

Anstatt wie im vergangenen Abschnitt Ableitungsfolgen durch die darin enthaltenen Variablen darzustellen, können wir dies ebensogut durch Folgen von Regeln im Kontext anderer Regeln tun.

Zum Beispiel könnte dann ein Modell der Ableitungen wie in 5.3 aussehen – ein Bi-Gramm-Modell von Regeln und Töchtern von Regeln:

$$P(\text{Parse}_i) = \prod_{R_j \in \text{Parse}_i} P(R_j | Dtr_k(R_l)) \tag{5.3}$$

Trainiert werden kann so ein Modell z.B. durch einfaches Abzählen über eine Baumbank.

Diese Vorgehensweise hat zunächst angenehme Eigenschaften:

- Die Vorgehensweise ist absolut generell. Von speziellen Eigenschaften von Merkmalsstrukturen bzw. des Formalismus, in dem die Merkmalsstrukturen geschrieben sind, wird abstrahiert. Alle bekannten Unifikationsgrammatiken sind daher in einem solchen Modell fassbar, da sie alle in Lexikonproduktionen[23] und Regelobjekten organisiert sind. Die Regelobjekte selbst sind natürlich immer endlich viele, weil von den Ausprägungen der MS abstrahiert wird.

- Die Fähigkeit eines solchen einfachen Modells, linguistische Generalisierungen zu adaptieren, hängt natürlich von der Spezialisierung der Regeln der Grammatik ab. Je spezieller die Regeln bezüglich "linguistischer Konstruktionen", desto besser greift das Modell. Dies ist offensichtlich, da das Modell nur Regelpaare beobachtet. Wenn Regelpaare "Konstruktions"-spezifisch sind, kann das Modell dies adaptieren. Falls die Regeln nur abstrakte Regelschemata sind, lernt das Modell (entsprechend der einhergehenden Parameterreduktion) wenig spezifische Regularitäten.

Durch die inhärente Tendenz, für Grammatiken mit stärkerer Ausprägung der Regeln besser zu werden, ist ein Optimum für kontextfreie Grammatiken erreichbar. Für HPSG-artige Unifikationsgrammatiken mit wenigen Regelschemata ist das Modell schlechter.

5.2.2 Unifikationsgrammatiken mit Typen

Für Unifikationsgrammatiken, die keine Typen verwenden, sind oben zwei Möglichkeiten aufgezeigt worden, probabilistische Modelle über den Ableitungen zu handhaben.

Durch die Klassenbildung über Merkmalsstrukturen kann einige Information bezüglich des Inhalts von Merkmalsstukturen in das statistische Modell hinübergerettet werden. Das Trainieren von Regelpaaren hingegen wirkt hier nur, wenn Regeln sehr konkret sind.

Moderne Unifikationsgrammatiken verwenden — inspiriert von Pollard & Sag's (1987) Arbeit über HPSG [72] — mehr und mehr Typsysteme zur Kodierung grammatischen Wissens.[24]

[23] Vollformenlexika sind hier natürlich vorausgesetzt. Auf spezielle Lemmatisierungstechniken wird hier nicht weiter eingegangen.

[24] Typsysteme wurden in der Einführung kurz dargestellt.

In diesem Abschnitt betrachten wir typisierte MS in der Art von Carpenter 1991 [14]. Hierbei stellt sich die Frage nach der Klassenzuordnung von MS in einer modifizierten Form.

Typisierte MS besitzen bereits einen Namen der als Klassenname verwendet werden kann und — wie wir im folgenden argumentieren — verwendet werden sollte.

5.2.2.1 PTUG: Bi-Gramm Modellierung von Typeshifts.

Die im vorangegangenen Abschnitt 5.2.1 erläuterte Vorgehensweise, wie etwa das Verfahren in Definition 5.3 diente dazu, Klassenzugehörigkeit eindeutig zu machen. Aus diesem Grunde wurde gefordert, daß die *Restrictions* in Vefahren 5.3 nicht unifizierbar miteinander sein sollten.

Typen sind aber – und das ist ihre wichtigste Eigenschaft – miteinander unifizierbar. Ein "flacher" Typverband, in dem alle Typen außer mit *Top* nicht mit anderen unifizierbar sind, ist ein Spezialfall, der z.B. eine Grammatik mit "Typ-Backbone" zu einer annotierten CFG werden läßt. Eine solche Grammatik nutzt die Mächtigkeit eines Typsystems nicht aus.

Durch die Verwendung von Typen als Backbone einer Unifikationsgrammatik entsteht also wieder das Problem der Klassenzugehörigkeit.

In der Unifikationsgrammatik, die in den Experimenten dieser Arbeit verwendet wurde, werden bereits aus Effizienzgründen, die in Abschnitt 3.2.4.2 erläutert wurden, die globalen Typen von MS als (eben nicht) kontextfreies Rückrat der Grammatik verwendet. Das folgende Beispiel erläutert anschaulich das Problem der Klassenzuordnung.

Die Grammatik enthält Regelschemata, die verschiedene Satztopologien des Deutschen beschreiben. Beispielsweise die folgenden Regeln in 5.4 und 5.5, wobei hier nur die Typnamen geschrieben werden und die MS als [..] erscheinen.

$$\text{S2[..]} \rightarrow \text{Vorfeld[...] Verbzweit[..] Mittelfeld[..]} \tag{5.4}$$

$$\text{S2[..]} \rightarrow \text{Vorfeld[...] Verbzweit[..] Mittelfeld[..] Verbletzt[..]} \tag{5.5}$$

Zusätzlich enthält die Grammatik Typdefinitionen wie in Beispiel 5.6:

$$\text{Vorfeld} > \text{PersPron Np SatzAdverb...} \tag{5.6}$$

Auf diese Weise wird sehr viel benutzerdefinierte grammatische Information durch Typdefinition ausgedrückt.[25]

Falls wir naiv die Typnamen als Klassennamen verwenden — wie in einer PCFG — laufen wir in folgendes Problem: Es gelte in der Typhierarchie $A > B\ C$ und nichts anderes. Es gilt dann per Definition (von PCFGs) $\sum_\beta P(A[..] \rightarrow \beta) = 1$, $\sum_\beta P(B[..] \rightarrow \beta) = 1$ und $\sum_\beta P(C[..] \rightarrow \beta) = 1$. Wenn nun in einer Ableitung eine MS vom Typ A vorliegt, dann gilt, da A mit MS vom Typ A, B oder C unifiziert werden kann: $\sum_\beta P(A[..] \rightarrow \beta) = 3$ — ein Widerspruch.

Offensichtlich ist die direkte Behandlung der Typen wie Variable einer PCFG nicht korrekt. Vielmehr ist die korrekte Sichtweise, daß ein Typ A in einer Ableitung

[25] Diese Verwendung von Typen könnte natürlich ebenso durch Grammatikregeln erledigt werden. Sie ist jedoch sehr viel ökonomischer in der Beschreibung und der Verarbeitung.

mit einem Typ B unifiziert wird *oder* daß ein Typ A in einer Ableitung mit einem Typ C unifiziert wird *oder* daß A mit einem Typ A unifiziert wird. Das heißt, daß für eine gegebene Grammatik und ein gegebenes Korpus eine Verteilung existiert, mit der A mit B oder C oder A unifiziert wird. Diese Verteilung kann wieder durch ein probabilistisches Modell beschrieben werden.

Für eine solche Vorgehensweise benötigen wir ein paar griffige Definitionen, die uns die Beobachtung des Verhaltens der Typen während einer Ableitung einer Unifikationsgrammatik erleichtern.

Definition 5.4 *Wir nennen ein Paar von Typen* **Typeshift**.

Definition 5.5 *Für eine typisierte Unifikationsgrammatik mit Regelmenge R der Form*

$$T_1[..] \rightarrow T_2[..]..T_k[..], T_l[..] \rightarrow T_m[..]..T_z[..], ... \tag{5.7}$$

nennen wir alle T_i **Regeltypen**.

Die Regeltypen sind diejenigen Typen, die wirklich in den linken und rechten Seiten der ursprünglichen Regeln der Grammatik auftauchen – also noch bevor eine Unifikation stattgefunden hat. Wir verwenden diese Teilmenge aller Typen, um die MS innerhalb von Regeln zu klassifizieren. Außerdem eignen sie sich, um bei der Behandlung von Typen die Anzahl zu trainierender Parameter klein zu halten.

Definition 5.6 *Gegeben eine Ableitung einer Unifikationsgrammatik G*

$$S[..] \stackrel{*}{\Rightarrow} x\alpha' y \Rightarrow xa\beta' by \Rightarrow xa\gamma' by \stackrel{*}{\Rightarrow} w \tag{5.8}$$

wobei β' erzeugt wurde aus β'' auf der rechten Seite einer Regel $\alpha \rightarrow a''\beta''b''$ unifiziert mit β, einer MS auf der linken Seite einer Regel $\beta \rightarrow \gamma$. Wir nennen das Paar von Regeltypen type(β''),type(β) einen **beobachteten Typeshift**.

Beobachtete Typeshifts sind also Paare von Regeltypen, die bei einer Ableitung beobachtet werden können, insofern als durch jede Regelanwendung auf die entsprechenden Regeltypen der beteiligten Regel geschlossen werden kann.

Beim probabilistischen Modell für die typisierte Unifikationsgrammatik gehen wir nun wie folgt vor: Einen Schritt einer Linksableitung der Grammatik zerlegen wir in alternierende Paare von Typeshift und Regel wie in 5.9:

$$S'[..] \stackrel{(S,A)}{\Rightarrow} A'[..] \stackrel{A[..]\rightarrow B[..]\beta}{\Rightarrow} B'[..]\beta' \stackrel{(B,C)}{\Rightarrow} C'[..]\beta'.... \tag{5.9}$$

Die obige Ableitung startet mit einem Startgraphen[26] S[..]. S wird durch die Unifikation von S[..] mit A[..] zu S' und A wird zu A'. In einer normalen Schreibweise für Linksableitungen wären S' und A' durch ein und dasselbe Symbol dargestellt. Sie sind auch bei unserer Sichtweise identisch. Wir trennen sie jedoch in der Darstellung, um ihre Historie zu verdeutlichen. S' bzw. A' ist das Resultat einer Unifikation eines Typs S und eines Typs A. Diese Tatsache schreiben wir als Typeshift (S,A), wobei S der Regeltyp des Startgraphen und A der Regeltyp der linken Seite der Regel $A[..] \rightarrow B[..]\beta$ ist.

Die obige Betrachtungsweise mag zunächst etwas kryptisch anmuten, dennoch bietet sie einige Vorteile:

[26]Startgraph ist diejenige MS einer Unifikationsgrammatik, die die Funktion des Startsymbols einer Phrasenstrukturgrammatik innehat.

- Indem wir die Klassen von MS bei Unifikationen über die Regeltypen bestimmen und nicht über die Typen der Ergebnis-MS, können wir für Regelanwendungen immer sofort den beteiligten Typeshift bestimmen. Bei einer Grammatik mit Koreferenz kann nämlich der Ergebnistyp erst am Ende eines ganzen Parses feststehen, da er durch andere Regelanwendungen bei Koreferenzen noch weiter verändert werden könnte.

- Bei einer lokalen Unifikation (also ohne Nebenwirkungen durch Koreferenzen) zweier Typen ist der Ergebnistyp deterministisch durch die Typhierarchie gegeben. In diesem Fall ist also die gesamte Information in den Argumenttypen einer Typunifikation enthalten.

Eine eigentlich dreistellige Relation stellen wir also durch eine zweistellige dar, weil in den meisten Fällen der dritte Wert feststeht und in allen anderen Fällen lokal nicht berechenbar ist. Im letzteren Falle ist zumindest garantiert, daß der Ergebnistyp von den beiden Regeltypen des Typeshifts subsummiert wird.

Zudem verschafft uns diese Vorgehensweise eine erträgliche obere Schranke für die Anzahl zu trainierender Parameter. Diese ist gegeben durch die Anzahl der Grammatikregeln und Lexikoneinträge zuzüglich des Quadrats der Anzahl der Regeltypen. [27]

Für Typsysteme mit Typverband und Eignungsfunktion (Aproppriatness) im Sinne von Carpenter 1991 [14] sind rekursive Typdefinitionen möglich. In Systemen wie dem von Emele & Zajac 1990 [26], wo Typen direkt über volle Definitionen von MS definiert sind, kann dies zu unendlich großen Mengen von komplexen Typen führen. Unser Ansatz ist robust gegenüber rekursiven Typdefinitionen, da die oben definierten Regeltypen unabhängig vom Typverband nur aus der jeweiligen gegebenen Menge von Grammatikregeln entnommen wird und immer endlich groß ist, weil die Typeshifts die Ausprägungen vernachlässigen und nur die gegebenen Typnamen nachbilden.

5.2.2.2 Erstellung eines Modells GM für eine typisierte Unifikationsgrammatik.

Nachdem wir die Grundlage für die Klassenbestimmung C_{TPUG} von Beobachtungen während einer Ableitung einer typisierten probabilistischen Unifikationsgrammatik (TPUG) im vorangegangenen Abschnitt motiviert haben, befaßt sich dieser Abschnitt mit der technischen Umsetzung. Dabei wird nochmals eine Definition für C_{TPUG} angeboten und ein Trainingsverfahren auf der Basis des Algorithmus von Fujisaki et al 1991 [33] dargestellt.

Definition 5.7 C_{TPUG}:

Eine Regelanwendung während einer Ableitung einer TPUG, bei der die k-te Tochter von Regel R_X, $Dtr_k(R_X)$ mit der Mutter der Regel R_Y, $Mtr(R_Y)$ unifiziert wird, beschreiben wir als Sequenz von Beobachtungen:

$$R_X,\ (Ruletype(Dtr_k(R_X)), Ruletype(Mtr(R_Y))),\ R_Y \qquad (5.10)$$

wobei R'_X und R'_Y Schritte in einer Linksableitung von TPUG sind.

[27] Die Unterscheidung von Regeltypen und sonstigen Typen hat nichts mit der Unterscheidung von benutzerdefinierten Typen der Typhierarchie und komplexen Typen zu tun, die durch Unifikation der ersten entstehen können.

Wir benötigen nun ein geeignetes probabilistisches Modell, das solche Sequenzen erzeugen kann. Hierbei ist eine gewisse Freiheit gegeben. Es ist durchaus möglich, beliebig lange linke Kontexte solcher Sequenzen zu modellieren, etwa wie bei Magerman's HBGM ([59]). Wir exemplifizieren den Ansatz mit einem einfachen Bi-Gramm von Typen und Regeln, bzw. Paaren von Typen, um auf einfache Trainingsverfahren zurückgreifen zu können. Dies ist jedoch keineswegs zwingend.[28] Unser Modell $\mathrm{GM}_{\mathrm{TPUG}}$ zerfällt in zwei Bi-Gramm-Modelle:

$\mathrm{GM}_{\mathrm{Tsh}}$:
: Jedem Paar von Typen (A, B) wird eine Wahrscheinlichkeit zugeordnet, sodaß gilt: $\forall A,\ \sum_B P(B|A) = 1$

$\mathrm{GM}_{\mathrm{Rule}}$:
: Jedem Paar von Typ und Regel $(A, A[..] \rightarrow \beta)$ wird eine Wahrscheinlichkeit zugeordnet sodaß gilt: $\forall A,\ \sum_\beta P(A[..] \rightarrow \beta)|A) = 1$

Für die Bewertung einer Darstellung einer Linksableitung wie in 5.7 würden alternierend der Schritt von einem Typeshift zu einer Regel von $\mathrm{GM}_{\mathrm{Rule}}$ bewertet und der nächste Schritt von einer Regel zu einem Typeshift von $\mathrm{GM}_{\mathrm{Tsh}}$.

Für die übliche Darstellung einer Ableitung (bzw. beim Parsing mit der PUG) wird die Wahrscheinlichkeit einer Regelanwendung ausgedrückt als:

$$P(D_{\mathrm{left}}) = \prod_{r_i \in D_{\mathrm{left}}} P(r_i|\mathrm{Dtr}_k(r_{i-1})),\ D_{\mathrm{left}} = S \overset{*}{\Rightarrow} \alpha \overset{r_{i-1}}{\Rightarrow} \beta \overset{r_i}{\Rightarrow} \gamma \overset{*}{\Rightarrow} w \tag{5.11}$$

$$P(r_i|\mathrm{Dtr}_k(r_{i-1})) = P(r|\mathrm{Ruletype}(\mathrm{Mtr}(r_i)))*P(\mathrm{Ruletype}(\mathrm{Mtr}(r))|\mathrm{Ruletype}(\mathrm{Dtr}_k(r_{i-1}))) \tag{5.12}$$

Wir ziehen also beim Parsing die Bewertungen der beiden Modelle wieder zusammen und trivialerweise gilt dann:

$$\sum_B \sum_\beta P_{GM}(B|A) * P_{GM}(B \rightarrow \beta|B) = 1 \tag{5.13}$$

da für $\mathrm{GM}_{\mathrm{Tsh}}$ und $\mathrm{GM}_{\mathrm{Rule}}$ in Isolation gilt:

$$\sum_B P_{GM_{\mathrm{Tsh}}}(B|A) = 1 \tag{5.14}$$

$$\sum_\beta P_{GM_{\mathrm{Rule}}}(B \rightarrow \beta|B) = 1 \tag{5.15}$$

Denn ein und derselbe Typ A wird innerhalb des Modells für die Typeshifts und des Modells für die Regeln strikt getrennt. Eine Situation, in der für einen gegebenen Typ in einer Ableitung entweder eine Typshift oder eine Regel angewendet wird, ist ausgeschlossen.

Die Bi-Gramme können, sofern eine geeignete Baumbank vorliegt, wie normale Bi-Gramme durch Abzählen trainiert werden. In diesem Fall wird der Korpus der

[28] Für die Experimente konnten keine komplexeren probabilistischen Modelle verwendet werden, da die Testgrammatik und der Korpus, die zur Verfügung standen, klein waren.

Baumbank zunächst geparst und alle Parses, die nicht der Baumbank entsprechen, werden aussortiert. Danach werden – getrennt voneinander – die Typeshifts und Regelanwendungen über den verbleibenden Parses abgezählt.

Da die manuelle Erstellung von Baumbänken durch menschliche Experten schwierig ist, benutzen wir eine Variante des Trainingsverfahrens von Fujisaki et al. für PCFGs, wie es in Abschnitt 2.1.2 der Einführung beschrieben ist.

Die Veränderungen, die nötig waren, sind kurz erläutert:

- Zunächst behandelten wir die Typeshifts wie unäre Regeln der Grammatik, deren Anwendung jedoch nicht mit denen von anderen Regeln konkurriert.[29] Dadurch können wir beide Modelle auf einfache Weise unüberwacht zusammen trainieren. Lexikoneinträge in L_{TPUG} werden ebenfalls wie unäre Regeln behandelt, konkurrieren aber mit Regeln aus R_{TPUG} desselben Regeltyps der linken Seite.

- Fujisaki's Verfahren ist eigentlich für Grammatiken in CNF gedacht. Beim Berechnen des relativen Counts für Regeln, werden die absoluten Counts gewichtet durch die relativen Wahrscheinlichkeiten der Parses, in denen sie vorkommen. Diese Gewichtung ist nur dann gerecht, wenn alle Bäume aus gleich vielen Regeln bestehen, was garantiert ist, wenn nur binär verzweigende Regeln verwendet werden, nicht aber in anderen Fällen. Um dies zu beheben, und verschieden lange Ableitungssequenzen gleich zu behandeln, ersetzten wir in diesem Schritt die relative Wahrscheinlichkeit eines Parses durch die relative Wahrscheinlichkeit eines Parses pro Regel, um ein längenunabhängiges Maß für die Bewertung einer Ableitung und den Regeln, die in dieser vorkommen, zu erhalten.

Der gesamte Algorithmus, wie er von uns verwendet wurde, ist in Definition 5.8 gegeben:

Definition 5.8 *Fujisakis Algorithmus für GM_{TPUG}:*

1. *$TSH := \{(A,B)|A,B$ sind unifizierbar, A ist Regeltyp einer MS einer rechten Regelseite, B ist Regeltyp einer linken Seite.$\}$*
2. *$RULES := \{r|r \in R_{TPUG}\} \cup \{A[..] \to w|(w, A[..]) \in L_{TPUG}\}$*
3. *Parse einen Korpus $\{B^i\}$*
4. *D^i_j sei die j-te Ableitung von B^i, dargestellt als Multimenge[30] von Regeln $r \in RULES$ und Typeshifts $tsh \in TSH$, die in D^i_j vorkommen.*
5. *Initialisiere zufällige Wahrscheinlichkeiten, sodaß Gleichungen 5.13 bis 5.15 gelten.*
6. *Berechne die $P(D^k_j)$ wie in Gleichungen 5.11 und 5.12.*
7. *Berechne die $NormP(D^k_j)$ als $P(D^k_j)^{\frac{1}{n}}$, wobei n die Anzahl der Regeln und Typeshifts in D^i_j ist.*

[29] Üblicherweise nennt man das Umbenennung.
[30] Multimengen sind Mengen, die Elemente mehrmals enthalten können.

8. *Berechne einen nach relativen Wahrscheinlichkeiten der Ableitungen gewichteten Zähler $CR^i_A(\beta)$ für die Regelanwendungen von $A[..] \rightarrow \beta$, wobei $count^i_j(A[..], \beta)$ die absolute Anzahl der Regelanwendungen $A[..] \rightarrow \beta$ in Ableitung D^i_j ist.*

$$CR^i_A(\beta) = \sum_j \left(\frac{NormP(D^i_j)}{\sum_k NormP(D^i_k)} count^i_j(A[..], \beta) \right) \tag{5.16}$$

9. *Berechne analog den gleichen gewichteten Zähler $CT^i_A(B)$ für die Typeshifts von A nach B, wobei $count^i_j(A, B)$ die absolute Anzahl der Typeshifts (A, B) in Ableitung D^i_j ist.*

10. *Normalisiere die Zähler CT und CR getrennt, zuerst über alle Regeln (bzw. Typeshifts) des selben Regeltyps und dann über alle Sätze der Stichprobe, in der der Regeltyp in einem Parse vorkam. Beachte immer nur solche Sätze der Stichprobe für einen Typ A, für die ein D^i_j existiert, sodaß A in D^i_j vorkommt.*

$$f_A(\beta) = \frac{\sum_{i, \alpha \in D^i_j} \frac{CR^i_\alpha(\beta)}{\sum_\gamma CR^i_\alpha(\gamma)}}{\sum_{i, \alpha \in D^i_j} 1} \tag{5.17}$$

$$f_A(B) = \frac{\sum_{i, \alpha \in D^i_j} \frac{CT^i_\alpha(\beta)}{\sum_\gamma CT^i_\alpha(\gamma)}}{\sum_{i, \alpha \in D^i_j} 1} \tag{5.18}$$

11. *Ersetze die Parameter $P_{rule}(A[..] \rightarrow \beta | A)$ durch $f_A(\beta)$. Ersetze die Parameter $P_{tsh}(B|A)$ durch $f_A(B)$.*

    Gehe wieder zu Schritt 6 und brich ab, wenn in einer Iteration keine Veränderung der Parameter mehr stattfindet.

Wir behandeln diejenigen Typeshifts, die nie beobachtet werden können, weil die Typen nicht miteinander unifizierbar sind, indem wir ihnen die harte Wahrscheinlichkeit 0 geben. Mögliche Typeshifts, die unifizierbar sind, aber keine beobachteten Typeshifts sind, müssen geglättet werden (Smoothing). Hierfür gibt es Standardtechniken für Bi-Gramme.

In unserer Testgrammatik, einer erweiterten Version der Grammatik, die in Abschnitt 4 für Experimente verwendet wurde, haben wir, da nicht alle Trainingssätze voll geparst werden konnten, diejenigen Wörter des Lexikons, die in keinem Parse vorkammen, durch eine einfache Smoothingtechnik behandelt. Durch die von der Grammatik vorgegebenen Typen der Lexikoneinträge ist eine Klassendefinition auf Wörtern gegeben. Diejenigen Wörter, die auf Wahrscheinlichkeit 0 gesetzt wurden, erhielten den Wert des schlechtesten Klassenvertreters, anschließend wurden alle Klassenvertreter so abgesenkt, daß die Summe wieder 1 entsprach. Bezüglich der Perplexitäten ist dieses Verfahren sehr schlecht, da es zu einer starken Glättung der durch das Modell beschriebenen Verteilung führt. Die 200 Trainingssätze der Zugauskunftsdomäne bilden jedoch einen recht kleinen Trainingskorpus von dem nicht alle Sätze geparst werden konnten und die starke Glättung wirkt hier einer Überadaption entgegen. Über den Testsätzen, die in Abschnitt 5.3.2 verwendet wurden erreichte die Grammatik eine Testsatzperplexität von 2.3 ohne, bzw. 4.57 mit Lexikonglättung für die Erzeugung der jeweils besten Ableitung.

Diese Werte sind nicht direkt vergleichbar mit einer N-Gramm Perplexität, die String- und nicht Ableitungsbezogen ist. Strings sind im allgemeinen kürzer als Ableitungen, die sie erzeugen. Für unserer Testsätze war das Verhältnis etwa 1 zu 2,5. Wenn also in die Kalkulation der Testsatzperplexität nicht mit $\frac{1}{\text{length}(D)}$ sondern $\frac{1}{\text{length}(S)}$, potenziert wird (S sei der Eingabestring), erhalten wir eine Abschätzung der Einschänkung auf die Wortkette. Hier liegen die Werte bei 6.95 ohne und 34.58 mit Lexikonglättung, im Verhältnis zu unserem Bi-Gramm-Modell mit TSP von 54.28.

Generell liegt, wie sich in den Experimenten zeigen wird, ein noch größerer Vorteil in einem probabilistischen Modell der Unifikationsgrammatiken. Durch die unmöglichen Typeshifts, werden extrem viele Unifikationen beim Pruning verworfen, die sowieso bereits durch Typunverträglichkeit scheitern würden.

In diesem Abschnitt 5.2 sind verschiedene Vorgehensweisen beschrieben worden, wie probabilistische Unifikationsgrammatiken behandelt werden können. Dabei wurde Bezug genommen auf bestehende Ansätze für Grammatiken auf ungetypten Formalismen, wie sie spärlich in der Literatur bereits behandelt wurden. Wir bemühten uns um eine systematische Überblick und eine Diskussion der sich ergebenden Aspekte und Möglichkeiten des Parsing.

Eine neue Vorgehensweise für Grammatiken auf typisierte Formalismen wurde dargestellt. Diese fußt natürlich in ihrer Wirkung stark auf der speziellen Verwendung der globalen Typen von MS in Regeln als Träger der sogenannten *Part-of-Speech*-Information. Die Experimente im Abschnitt 5.3.2 sind mit einer solchen Grammatik durchgeführt worden.

In HPSG-artigen Grammatiken ist *Part-of-Speech*-Information üblicherweise tief verschachtelt und in Werten mehrer Merkmale dargestellt. Dies macht eine Kopplung natürlich schwierig.

Eine Übersetzung in ein anderes Regelformat scheint uns die einzig plausible Lösung zu sein, dieses Problem anzugehen. Dabei würden – ähnlich dem Ansatz zur Klassenbildung bei typfreien Unifikationsgrammatiken – aus allen Kombinationen der Werte der relevanten Merkmale neue Klassennamen gebildet. Diese Klassennamen können schließlich entweder dynamisch – per *Restriction* beim Parsing und Training abgefragt werden. Oder aber die Typdefinition wird so verändert, daß sie als neue jeweils unverträgliche Subtypen des globalen Regeltyps *sign* in die Typdefinition aufgenommen werden und entsprechend viele Kopien der Regelschemata gebildet werden. Beide Wege sind mühselig und sicher nicht effizient realisierbar. Der Vollständigkeit halber sind diese Möglichkeiten hier dennoch erwähnt, obwohl wir HPSG für die in dieser Arbeit entwickelten Verfahren für nicht geeignet halten.

5.3 LR-ACP mit probabilistischer Unifikationsgrammatik

Dieses Kapitel widmet sich der Erweiterung der Metrik des LR-ACP, wie sie in Abschnitt 3.2.6.2 vorgestellt wurde. Darüber hinaus wird die Einbettung der TPUG in den LR-ACP beschrieben – also Aspekte, die über das reine Bewerten von Kanten und Paaren von Kanten hinausgehen.

5.3.1 Eine einheitliche Metrik für den LR-ACP

In Abschnitt 3.2.6.2 wurde für den LR-ACP eine Methode angegeben, Kanten und Paare von Kanten auf der Agenda zu bewerten. In diesem Abschnitt gehen wir etwas genereller vor, indem wir annehmen, wir hätten n verschiedene probabilistische Modelle, die eine Kante bewerten. Hierfür sind dann n-1 Gewichte nötig, um die Bewertungen der einzelnen Modelle in Bezug zu setzen. Obwohl genereller, ist diese Beschreibung der Bewertungsfunktion doch implementierungsnäher. Zunächst klären wir, wie für die Aufteilung einer Kantenbewertung in INSIDE-Bewertung und OUTSIDE-Bewertung, bzw. für die Operationen zur Normalisierung eines Modells vorgegangen wird. Da wir eine strikte Viterbi-Bewertung verfolgen, bietet sich, wie in Abschnitt 3.2.6.2 die Verwendung von logarithmisierten Werten an.[31]

Definition 5.9 *Eine Bewertungskomponente für eine probabilistisches Modell M für eine Kante K ist ein Record Rec^M mit 4 Feldern:*

IS *: Enthält $LogP(K.S_M|M)$, die INSIDE-Wahrscheinlichkeit für K bezüglich M.*

OS *: Enthält $LogP(Leftcontext(K.S_M), K.S_M|M)$, die OUTSIDE-Wahrscheinlichkeit für K bezüglich M.*

IO *: Enthält $Length(K.S_M)$, die INSIDE-Operationen für K bezüglich M.*

OO *: Enthält $Length(K.S_M) + Length((Leftcontext(K.M))$ die OUTSIDE-Operationen für K bezüglich M.*

wobei $K.S_M$ die für Modell M relevante Beobachtungssequenz S ist, die von K überspannt wird.

S kann dabei je nach Modell verschieden sein — etwa eine Wortfolge oder eine grammatische Ableitung.

Für n Modelle erhält eine Kante n Records $Rec^{M_1}...Rec^{M_n}$

Aus der Definition 5.9 folgt, daß $Rec^M.IS$ und $Rec^M.IO$ sofern K vorliegt, sofort berechnet werden kann.

Leftcontext$(K.S_M)$ ergibt sich dynamisch beim Parsing und wird durch SEEKDOWN Operationen von nicht leeren aktiven Kanten zu neu eingefügten leeren Kanten vererbt.

[31] Bei Vorwärts-Rückwärts-Bewertungen werden Wahrscheinlichkeiten addiert, was bei logarithmischer Darstellung kompliziert ist. Bei Viterbi-Bewertungen werden Wahrscheinlichkeiten nur multipliziert – entsprechend in logarithmischer Darstellung addiert.

Definition 5.10 *Wenn K keine terminale Kante ist, dann gilt: Leftcontext($K.S_M$) ist die für M relevante Beobachtungssequenz, die durch die beste Sequenz von Kanten $K_1..K_m$ überspannt wird, sodaß $begin(K_1) = 0$, $end(K_i) = begin(K_j)$ und $end(K_m) = begin(K)$ Wenn K eine terminale Kante ist, dann ist Leftcontext($K.S_M$) leer.*

Der Kontext, in dem eine Kante eingeführt wird, hängt immer von der besten Sequenz von aktiven Kanten ab, die vom Zeitpunkt 0 bis zum Anfangszeitpunkt dieser Kante reichen. Nur bei terminalen Kanten ist dies nicht der Fall, diese hängen von dem Modul in einem System ab, daß die Eingabehypothesen produziert. Wir sagen hier absichtlich nicht "Worthypothesen", da die Metrik auch bei anderen Granulaten als Wörtern funktionieren soll, also z.B. Phoneme oder andere Wortuntereinheiten.

Beim Erzeugen von neuen Kanten K werden allen $Rec^{M_i}(K)$ neue Werte zugewiesen. Die drei Fälle, die zu unterscheiden sind, sind:

- Initialisierung von terminalen Kanten und aktiver Startkante.
- Kanten als Ergebnis der fundamentalen Regel.
- Kanten als Ergebnis einer SEEK-DOWN Operation.

Die Initialisierungen werden nicht explizit angegeben. Es sei angemerkt, daß dies unnötig ist, da die Vorgehensweise bereits voll aus Definition 5.9 und 5.10 folgt: Da beim Einfügen der Startkante noch keine andere Kante existiert, ist der linke Kontext leer. Die Startkante überspannt kein Frame und kein Wort, jedoch eine Grammatikregel, erhält also von den jeweiligen Modellen die entsprechenden Bewertungen und Operationen. Entsprechend verhält es sich mit terminalen Kanten. Per Definition ist ihr linker Kontext leer (siehe 5.10), daher wird von allen Modellen keine OUTSIDE-Bewertung erzeugt[32]. Die OUTSIDE-Operationen sind 0.

Definition 5.11 *Kante A wird von der aktiven Kante B und der inaktiven Kante C eingeführt durch die fundamentale Regel:*

$$\begin{array}{llll} Rec^{M_i}.IS(A) & = Rec^{M_i}.IS(B) & +Rec^{M_i}.IS(C) & +LogP(Trans^{M_i}(B,C)|M_i) \\ Rec^{M_i}.OS(A) & = Rec^{M_i}.OS(B) & +Rec^{M_i}.IS(C) & +LogP(Trans^{M_i}(B,C)|M_i) \\ Rec^{M_i}.IO(A) & = Rec^{M_i}.IO(B) & +Rec^{M_i}.IO(C) & +Length(Trans^{M_i}(B,C)) \\ Rec^{M_i}.IO(A) & = Rec^{M_i}.IO(B) & +Rec^{M_i}.IO(C) & +Length(Trans^{M_i}(B,C)) \end{array} \tag{5.19}$$

entsprechend beschreiben wir die Einführung von leeren aktiven Kanten.

Definition 5.12 *Die leere aktive Kante A wird von der aktiven Kante B durch eine SEEK-DOWN Operation eingeführt:*

$$\begin{array}{llll} Rec^{M_i}.OS(A) & = Rec^{M_i}.OS(B) & +Rec^{M_i}.IS(A) & +LogP(Trans^{M_i}(B,A)|M_i) \\ Rec^{M_i}.OO(A) & = Rec^{M_i}.OO(B) & +Rec^{M_i}.IO(A) & +Length(Trans^{M_i}(B,A)) \end{array} \tag{5.20}$$

[32] In diesem Fall per Definition 0.

$Rec^{M_i}.IS(A)$ und $Rec^{M_i}.IO(A)$ werden hier wie in Definition 5.9 behandelt. Im übrigen gilt dies auch in Definition 5.11. Dort ist die Behandlung der Bewertungen generell äquivalent zur Definition 5.9. Dennoch ist es sinnvoll, diese Definitionen noch einmal anzugeben. Denn in 5.12 und 5.11 können wir auf einfache Weise die Schnittstelle zu spezifischen Eigenschaften der Modelle beschreiben. Die mit Trans(X,Y) bezeichnete Operation für zwei Kanten, von denen die linke eine aktive sein muß, ist in ihrer Definition spezifisch für die einzelnen Modelle. Falls M_i z.B. das akustische Modell bezeichnet, ist LogP(Trans(X,Y))= 0 und Length(Trans(X,Y)) = 0. Beim Kombinieren von akustischen Bewertungen zweier Kanten gibt es keine "akustische Transitionsstrafe", und die Länge der Beobachtungssequenz zur Berechnung der Übergangsstrafe ist ebenfalls 0.

Definition 5.13 *$Trans^{M_i}(B,A)$ ist die Sequenz von durch M_i erzeugten Beobachtungen beim Übergang von $B.S_{M_i}$ zu $A.S_{M_i}$*

Wenn A die Wortkette "ich will" und B die Wortkette "ein Bier" überdeckt, dann ist für M_i, sofern es sich um ein Bi-Gramm handelt, Trans(B,A)= [(will,ein)], sofern es sich um ein Tri-Gram handelt hingegen, Trans(B,A)= [(ich,will,ein),(will,ein,Bier)].

Für einzelne Kanten werden keine anderen Bewertungen berechnet. Für Paare von Kanten gilt:

Definition 5.14 *Bewertung eines Paares (A,B) von aktiver und passiver Kante auf der Agenda:*

$$AgendaScore(A,B) = \frac{rec^{M_1}.OS(C)}{rec^{M_1}.OO(C)} + \sum_{i=2}^{n} \gamma_i \frac{rec^{M_i}.OS(C)}{rec^{M_i}.OO(C)} \tag{5.21}$$

wobei C durch Anwendung der fundamentalen Regel aus A und B erzeugt würde und γ_i das Gewicht von M_i in Bezug auf M_1 ist.

Natürlich läßt sich die elegante Allgemeinheit wie oben nicht für die (ebenfalls wichtigen) Details der Realisierung eines Verarbeitungsmodells durchhalten.

In dieser Arbeit gehen nicht n, sondern drei probabilistische Modelle in die Bewertung von Kanten und Kantenpaaren auf der Agenda ein. Dennoch ist offensichtlich, daß ohne formale Schwierigkeiten noch andere Quellen probabilistischer Information hinzugefügt werden könnten. Hier kommen z.B. probabilistische Dialogmodelle in Frage – sodaß der linke Kontext der Startkante eben nicht leer wäre, wie oben angemerkt – oder aber Caches, im Sinne von Essen & Ney 1991 [27].

Auf der anderen Seite besitzt der LR-ACP für die Integration symbolischer Information auch die Schnittstelle der Startkante. Die Restriktionen von einer symbolischen Dialogkomponente in Form von MS kann über die Startkante in die Suche integriert werden. Hier ist noch ein weites Feld für Untersuchungen gegeben, die in dieser Arbeit nicht mehr unternommen werden.

Der LR-ACP mit TPUG betrieben, führt zu einigen Möglichkeiten, die in Abschnitt 4 vorgestellten Verfahren zu verändern.

Die Berechnung der Prädiktion, wie in Abschnitt 4.4, wird verbessert, indem zusätzliches Wissen den Anteil des Beamsearch verbessert. Der Vorschlagsschritt wird weiterhin nur mit dem Bi-Gramm-Modell ausgeführt, da sich die TPUG nicht in der gleichen Form auskompilieren läßt. Dies gälte auch bei einem GPLRP wie etwa von Briscoe & Carroll 1993 [13], modifiziert in der Art von Staab 1994 [89], da die entsprechenden Reduce-Anteile doch wieder mit ausgeführt werden müßten.

Die Bewertung, die sowohl bei TDPI, als auch bei BUITDV vom Parser an den Decoder gesendet wird, kann eine Kombination aus N-Gramm- und GM-Bewertung sein. Hier ist es jedoch schwierig – außer zur Auswahl des globalen Suchpfades, aufgrund dessen die Übergangsstrafe generiert wird – einen GM-Anteil auszuwählen, der über die Wahrscheinlichkeit des eigentlichen Lexikonzugriffs hinausgeht. Ein N-Gramm-Modell eignet sich aufgrund seines Formats besser zur Bewertung von Wortübergängen als eine probabilistische Grammatik.

Eine Bewertung kann dennoch wie folgt aussehen:

Definition 5.15 *Übergangsstrafe, die als Prädiktion oder Verifikation an einen Beamdecoder gesendet werden kann:*

$$\begin{aligned}&score = \\ &\gamma * logP(first(W.string)|last(A.intro), n\text{-}gram) + \\ &\delta * logP(type(W.mother)|type(A.next), gm) + \\ &\delta * logP(W.rule|type(W.mother), gm)\end{aligned} \tag{5.22}$$

wobei γ und δ Gewichte für den N-Gramm- bzw. Grammatikanteil sind. W ist die terminale Kante, aufgrund der eine Verifikation gesendet wird, bzw. aufgrund der eine Prädiktion ausgelöst wird. A ist die entsprechende aktive Kante mit W auf der Agenda.

Bei dieser Art von Übergangsstrafe wird die Regelwahrscheinlichkeit des Lexikonzugriffs und der Typeshift, der diesen Lexikonzugriff mit einer bestehenden partiellen Analyse verbindet, in die Transition mitaufgenommen.

Durch das Grammatikmodell werden auch die a priori unmöglichen Unifikationen bewertet. Diese werden **immer** beim Pruning verworfen, indem die Paare von Kanten, die eine Wahrscheinlichkeit 0 vom Grammatikmodell zugewiesen bekommen, nicht in eine Agenda einsortiert werden.

5.3.2 Experimente

Im letzten Abschnitt dieser Arbeit werden einige selektive Experimente vorgestellt. Dabei wird der LR-ACP wieder mit dem Decoder aus Abschnitt 4 gekoppelt.

Die akustischen Modelle die für die Experimente in diesem Abschnitt verwendet wurden, sind so adaptiert worden, daß sich über den Testsätzen für die beste Kette des Decoders eine Wortakkuratheit von 81,4 % ergab – also etwas schlechter als bei den einfachen Experimenten in Abschnitt 4.5. Es mußte also mit größeren akustischen Beamweiten gearbeitet werden, um zu garantieren, daß die richtigen Wortfolgen jeweils noch im Wortgraphen enthalten sind. Die auftretenden Wortgraphendichten schwanken zwischen 50 und 200.

Das verwendete Bi-Gramm-Modell *BM* ist das selbe wie das in Abschnitt 4 verwendete mit einer Testsatzperplexität von 54.28.

Die Grammatik, die hier verwendet wurde, ist größer als die in Abschnitt 4 verwendete. Insbesondere haben die MS wesentlich mehr Merkmale zum Strukturaufbau, enthalten jedoch weniger restriktive Merkmale.[33] Die Grammatik enthält 44 Regelschemata[34]. Im Lexikon sind 363 Wortformen vertreten mit einer durchschnitt-

[33] Was in unserer kleinen Testumgebung eine realistische Grammatik annähern soll.

[34] Die Grammatik in Abschnitt 4 enthält nur 31 Regelschemata.

lichen Ambiguität von 1.16.

Die Testsatzperplexität des Grammatikmodells *GM* wurde für die besten Ableitungen berechnet. Man beachte, daß sie sich auf Ableitungsbäume bezieht und nicht auf Ketten. Ableitungsbäume werden üblicherweise aus wesentlich mehr Regeln aufgebaut, als die Kette der erzeugten Terminale lang ist. Im Mittel besaßen die erzeugten Beobachtungssequenzen der besten Ableitungen eine Länge von 35. Indem wir bei der Berechnung der TSP beim Normalisieren über die Länge die jeweiligen Testsätze einsetzen, erhalten wir einen vagen Eindruck, wie sich die Beschränkung des Modells mit der des N-Grams vergleichen läßt.

TSP	Ohne Glättung	Lexikonglättung
Beste Ableitungen	2.3	4.6
Sätze	7	34.6

Im folgenden wurde mit allen Kopplungs-Varianten vergleichende Tests durchgeführt. Dabei wurden verschiedene Gewichtungen von Bi-Gramm-Modell und Grammatik-Modell verwendet. Zur Erinnerung sei erwähnt, daß γ das Verhältnis von *BM* und δ von *GM* zur akustischen Bewertung regelt.

Die folgende Tabelle ergibt die Ergebnisse für verschiedene Kombinationen von *BM* und *GM*. Hierbei ist gemittelt über alle drei Kopplungen BUI, BUITDV und TDPI, mit denen unter gleichen Bedingungen die gleiche Anzahl Tests auf den Testsätzen durchgeführt wurde. Alle Experimente wurden mit Time Mapping durchgeführt.

γ	δ	Tries	Prunes	Time	Rec	Rank
0.8	0.0	30674	15088	330.4	53.3	3.0
0.0	0.8	1928	123658	160.3	26.7	1
0.5	0.5	2104	72080	124.0	46.7	1.12
0.6	0.3	2522	60613	141.0	60.0	2.0

Hierbei sind

Tries die wirklich ausgeführten Completer-Operationen

Prunes die durch Pruning verworfenen Completer-Operationen

Time die CPU-Zeit, die nur vom Parser alleine verbraucht wurde in Sekunden.

Rec die Satzerkennungsrate bezüglich Syntax. Semantisch ähnliche Resultate, die also z.B. den gleichen Benutzerwunsch ausdrückten, wurden als nicht erkannt gezählt.

Rank der Rang des richtig erkannten Satzes, falls mehrere erkannt wurden.

Wie sind nun die in der Tabelle angegebenen Resultate zu interpretieren?

Zunächst natürlich mit der gegebenen Vorsicht bezüglich der Erkennungsraten. Diese dürfen aufgrund des beschränkten Skopus der Experimente nicht absolut gesehen werden. Wir sind hier jedoch vor allem am relativen Verhalten des LR-ACP interessiert und dies läßt sich aus den Daten gut ablesen.

Zunächst wollen wir die Haupttendenzen benennen, die wir aus den Ergebnissen schließen:

1. Das probabilistische Grammatikmodell kann das Bi-Gramm-Modell nicht ersetzen.

2. Das Bi-Gramm-Modell alleine ist schlechter als eine Kombination aus Bi-Gramm-Modell und Grammatik-Modell.

3. Die Kombination von Bi-Gramm-Modell und Grammatikmodell ist besser, wenn das Bi-Gramm stärker gewichtet ist.

Diese Schlüsse sind zunächst vage, lassen sich jedoch begründen.

Die Schlussfolgerung 1 ist sehr deutlich an den Messwerten abzulesen. Die Erkennungsrate für das Grammatikmodell alleine ist gerade halb so gut wie für das Bi-Gramm alleine. Dies läßt sich nicht nur in Perplexitäten erklären[35]. Wenn nur das Bi-Gramm alleine verwendet wird, steht immer noch die symbolische Beschränkung der Unifikationsgrammatik zur Verfügung, die einem Modell *GM* mit gleichmäßigen Verteilungen für alle rechten Regelseiten und Typen entspräche. Wenn jedoch nur das Grammatikmodell verwendet wird, ist die lokale Beschränkung auf Wortketten sehr gering. Durch die Glättung des Lexikons verschwindet sie fast. An den Ergebnissen bezüglich des Rangs können wir jedoch ablesen, daß, wenn die richtige Wortkette den Beamsearch überlebte, die entsprechende Analyse auch auf Rang 1 gelangt. Bei den vielen falsch erkannten Äußerungen des *GM* alleine gab es einige korrekte Analysen falscher Wortketten. Durch eine Kette von Wörtern können sehr viele Analysen entstehen. Ohne das N-Gramm-Modell ist der Suchraum also um Größenordnungen weiter. Die Beschränkung durch ein Modell auf Wortsequenzen kann also offensichtlich nicht ersetzt werden.

Die Schlussfolgerung 2 ist ebenfalls deutlich an den Werten abzulesen. Die Erkennungsrate ist zwar einigermaßen gut, der Rang ist jedoch schlecht. Wortverwechslungen, die lokal plausibel sind – also vom Bi-Gramm gut bewertet, werden erst durch den größeren Kontext des Grammatikmodells wieder schlechter bewertet. Hierzu ist zu sagen, daß die Unifikationsgrammatik, die für die Tests verwendet wurde, nicht sehr restriktiv war. Durch das Grammatikmodell erst, das ja korpusbasiert ist[36], werden durch Übergenerierung der Grammatik erzeugte Ableitungen von sinnlosen Sätzen abgeschwächt.

Bezüglich der reinen Erkennungsleistung war 2 im Grunde zu erwarten. Wann immer ein Modell hinzugefügt wird, dessen Perplexität niedriger ist als die Anzahl möglicher nächster Beobachtungen, steigt die Erkennungsleistung. Um so verwunderlicher, daß die Erkennung der gleichgewichteten Kombination schlechter ist, als das Bi-Gramm alleine. Wie ist dies zu erklären?

Der Grund hierfür ist unsere Zählweise von "erkannt" vs "nicht erkannt". Als erkannt gilt nur, wenn die richtige Kette und Analyse vorliegen. Im Falle der gleichgewichteten Kombination traten Fälle auf, bei denen der LR-ACP Wörter einfügte, die keine semantische Veränderung zur Folge hatten[37].

Im Hinblick auf Robustheit können solche Fälle jedoch nicht als Erfolge gezählt werden, da ja z.B. die Verwechslung eines Inhaltswortes der gleichen Gattung[38] dann

[35] Unser Bi-Gramm war sowieso stark geglättet und hatte keine besonders gute TSP.

[36] Im Korpus waren natürlich keine sinnlosen Sätze

[37] Das schönste Beispiel war ein "Bitte" am Ende eines Satzes, welches durch das höfliche Programm eingeführt wurde, jedoch nicht vom ungehobelten Sprecher geäußert war.

[38] Z.B. zwei Zielorte der Eisenbahndomäne, wie Nürnberg und München.

auch gezählt werden müßte. Wenn durch das Bi-Gramm alleine jedoch die richtige Kette gefunden wurde, so wurde – neben allen anderen möglichen Ableitungen – auch die richtige Ableitung gefunden. Wir müssen hier also die Abhängigkeit von Rang und Erkennung berücksichtigen, um einen richtigen Eindruck zu erhalten.

Mit einer gewichteten Kombination, bei der das Bi-Gramm überwiegt, wird gegenüber der reinen Bi-Gramm Variante eine echte Verbesserung erzielt, sowohl bei der Erkennung, als auch bezüglich des Rangs. Dies führt uns zur obigen Schlussfolgerung 3.

Bei einer Kopplung, wie sie in dieser Arbeit vorgenommen wurde bleibt das Hauptproblem die Suche in der Menge der Worthypothesen. Diese wird durch die direkte Beschränkung auf Wortfolgen besser geleistet. Wir sehen an den Werten der *Tries* und *Prunes*, daß der Unterschied zwischen gleichgewichteter und zugunsten des Bi-Gramms gewichteter Kombination nicht groß ist. Dies spiegelt sich auch in den Laufzeiten wieder. Dennoch ist das Suchergebnis besser.

5.4 Konklusion

In der vorliegenden Arbeit ist eine enge zeitsynchrone Kopplung zwischen Analyse und Erkennung vorgenommen worden. Dabei stand im Vordergrund die Entwicklung eines speziellen Chartparsers für diese Zwecke. Wir sind nicht der Ansicht, daß die hier vorgeschlagenen Verfahren ultimativ sind. Sie zeigen jedoch Wege auf, wie von einer rein sequentiellen Architektur zu etwas stärker integrativen Architekturen gelangt werden kann. Dabei kann auf den reichen Schatz an Wissen aus den Bereichen Sprachverstehen und Mustererkennung zurückgegriffen werden.

Es ist nicht sinnvoll, das Rad immer wieder neu zu erfinden. Dem wurde versucht, hier Rechnung getragen, und so wird der hier präsentierte Ansatz in dem Moment besser, wenn in den klassischen Verfahren Verbesserungen erzielt werden. Er ist in sofern offen gegenüber besseren Grammatiken und besseren probabilistischen Modellen. Wenn in der reinen Spracherkennung z.B. N-Gramme mit größerem N zu besseren Perplexitäten führen, so profitiert dieser Ansatz ebenfalls hiervon.

Eines der großen Probleme des Ansatzes ist die flexible Strahlensuche. Diese muß in Zukunft noch besser auf Ableitungswahrscheinlichkeiten abgestimmt werden. Das Pruning einer einzigen Regel kann bereits zu einem totalen Fehlschlag des ganzen Parsing führen. Der präsentierte Ausweg des zweiten Suchzyklus, der nicht weiter experimentiert wurde, stellt keine ultima ratio dar, ist jedoch zunächst ein Ausweg.

Die Möglichkeit, eine enge Kopplung überhaupt vorzunehmen, hing im übrigen nicht wenig von einem gut und effizient verarbeitbaren Format der Grammatik ab. Für sehr mächtige Grammatiken, wie etwa volle HPSGs, die auf Inferenzmaschinen implementiert sind, wird sich dieser Ansatz daher wahrscheinlich als nicht tragfähig erweisen.

Sowieso scheint eine zu weitgehende "von Hand"-Kodierung der sprachlichen Beschränkungen in einer Grammatik nicht für den hier gegangenen Weg geeignet. Während eine "gute" Grammatik die Teile von Hand kodiert enthalten sollte, die für die Erzeugung einer logischen Form benötigt werden, sollten die Beschränkungen auf Wort- oder Phrasenfolgen gelernt werden. Dies ist gerade durch PUG, wie sie hier beschrieben wurden, möglich. Traditionelle Vorgehensweisen, wie z.B. HPSG, lassen ein Upscaling auf große Datenmengen oder spontane Sprache wahrscheinlich nicht zu.

Während also mit starken Beschränkungen eine enge Kopplung in dieser Arbeit geglückt ist, stellt sich die Frage, wie sich der LR-ACP beim Wegfall einiger dieser Beschränkungen verhalten wird.

- Die akustische Erkennung war in allen Experimenten bereits sehr gut. Beim Übergang auf spontane Sprache wird in absehbarer Zukunft nur noch mit Worterkennungsraten von 50 bis 60 % zu rechnen sein.
- Die Abdeckung der verwendeten Grammatik war klein. Der Suchraum steigt bei realistischen Grammatiken stark an.

Dennoch sind auch die möglichen Mittel noch lange nicht ausgeschöpft. In Zukunft kann ein Ansatz in der Art des LR-ACP noch um einiges verbessert werden. Hier bieten sich an:

- Prinzipiell sind im LR-ACP durchaus auch N-Gramm-Modelle mit größerem N einsetzbar.[39]
- Für die probabilistische Unifikationsgrammatik können ebenfalls Modelle verwendet werden, die einen größeren Kontext modellieren und somit die Perplexität verringern.
- Flexiblere Methoden zur Bestimmung von Suchstrahlen beim Parsing sind sicher nötig. In der Arbeit wurden einfache Beamschwellen aus dem Decoding verwendet. Hier sind speziellere Verfahren für Parsing zu entwickeln.

Zunächst bleibt aber ein ausgiebiges empirisches Testen integrativer Ansätze die Aufgabe.

[39] Obwohl dies zu größerem Buchhalterischem Aufwand führt, wie z.B. bei den Update-Operationen des Time Mappings.

Literaturverzeichnis

[1] Aho, A. und J. Ullman. *The Theory of Parsing, Translation and Compiling.* Band 1. N.J. Prentice Hall: Englewood Cliffs, 1972.

[2] Ait-Kaci, Hassan. *A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures.* Diss. University of Pennsylvania, USA, 1984.

[3] Altmann, Gerry und Henry Thompson. „Modularity Compromised". Altmann, Gerry, Hg., *Cognitive Models of Speech Processing*, MIT Press Cambridge, 1990. MIT Press.

[4] Amtrup, Jan. Parallele Strukturanalyse natürlicher Sprache mit Transputern. Diplomarbeit, Universität Hamburg, FB Informatik, AB NatS, März 1992.

[5] Andry, F. und S. Thornton. „A Parser for Speech Lattices Using a UCG Grammar.". *Proc. Eurospeech 1991*, Band 1, 1991. 219–222.

[6] Aubert, X., C. Dugast, H. Ney und V. Steinbiss. „Large Vocabulary Continuous Speech Recognition of Wall Street Journal Data.". *ICASSP*, 1994.

[7] Backofen, Rolf. Integration von Funktionen, Relationen und Typen beim Sprachentwurf. Diplomarbeit, Universität Erlangen Nürnberg, 1989.

[8] Baker, J.K. „Trainable Grammars for Speech Recognition". *Speech Communication Paper, 97th Meetingh of the Acoustical Society of America*, Cambridge, Mass., 1979. MIT Press.

[9] Baum, L. E., T. Petrie, G. Soules und N. Weiss. „A maximation technique occurring in the statistical analysys of probabilistic functions of Markov chains". *Ann. Math. Stat.*, (1970), 41:164–171.

[10] Bresnan, J. und D. Kaplan. *The Mental Representation of Grammatical Relations.* Cambridge, Mass.: MIT Press, 1982.

[11] Brietzmann, Astrid. „Reif Für die Insel. Syntaktische Anslyse gesprochener Sprache durch bidirectionales Chart-Parsing". Mangold, H., Hg., *Sprachliche Mensch Maschine Kommunikation.* Oldenburg Verlag, 1992. 103–116.

[12] Briscoe, Ted. *Modelling Human Sentence Processing: A Computational Approach.* Chichester: Ellis Horwood, 1987.

[13] Briscoe, Ted und John Carroll. „Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars“. *Computational Linguistics*, (1993), 19(1):25–59.

[14] Carpenter, Bob. *Typed Feature Structures: Inheritance (In)equality and Extensionality.* CMU, 1991.

[15] Charniak, E. und G. Carroll. „Context-Sensitive Statistics For Improved Grammatikcal Language Models.“. *National Conference of the American Association for Artificial Intelligence*, Seattle, 1994.

[16] Chien, Lee-Feng, K.J. Chen und Lin-Shan Lee. „An Augmented Chart Data Structure with Efficient Word Lattice Parsing Scheme in Speech Recognition Applications“. *Proceedings of COLING*, 1990. 60–65.

[17] Chien, Lee-Feng, Keh-Jiann Chen und Lin-Shan Lee. „A Best-First Language Processing Model Integrating the Unification Grammar and Markov Language Model for Speech Recognition Applications.“. *IEEE Transactions on Speech and Audio Processing*, Band 1,2, 1993. 221–240.

[18] Chow, Yen-Lu und Richard Schwartz. „The N-Best Algorithm: An Efficient Procedure for Finding Top N Sentence Hypothesis“. *DARPA Speech and Natural Language Workshop*, Cape Cod, Mass., October 1989. 199–202.

[19] Corazza, A., R. Gretter, G. Satta und De Mori R. „Computation of upper bounds for island-driven stochastic parsers.“. *Proceedings of the Eurospeech in Genova*, (September 24.-26. 1991), 1:215–218.

[20] Davis, H. W., A. Bramanti-Gregor und J. Wang. „The Advantages of Using Depth and Breadth Components in Heuristic Search“. Ras, Z. W. und L. Saitta, Hg., *Methodologies for Intelligent Systems.* Band 3. Amsterdam: North-Holland, 1989. 19–28.

[21] Dörre, Jochen und Andreas Eisele. „Determining Consistency of Feature Terms with Distributed Disjunction“. *Proceedings of 13th GWAI*, 1989. 270–279.

[22] Dörre, Jochen und Roland Seiffert. Sorted Feature Terms and Relational Dependencies. IWBS-Report 153, IWBS, Februar 1991.

[23] Dupont, Pierre. „Dynamic Use of Syntactical Knowledge in Continuous Speech Recognition.“. *Proc. Eurospeech 1993*, 1993. 1959–1962.

[24] Earley, J. „An Efficient Context-Free Parsing Algorithm“. *Comm. ACM*, (1970), 13:94–102.

[25] Eisele, Andreas und Jochen Dörre. Disjunctive Unification. IWBS-Report 124, IWBS, May 1990.

[26] Emele, Martin und Rémi Zajac. „A Fixed-Point Semantics for Feature Type Systems“. *Proc. of the 2nd International Workshop on Coditional and Typed Rewritung Systems (CTRS)*, Montreal, June 1990.

[27] Essen, Ute und Hermann Ney. „Statistical Language Modelling Using a Cache Memory“. *QUALICO 1st Quantitative Linguistics Conference*, Trier, September 1991. GLDV.

[28] Euler, Lutz. Eigenschaften von Unifikationsformalismen. Technischer Bericht ASL-TR-23-91/UHH, Universität Hamburg, Bodenstedtstraße 16, D-W-2000 Hamburg 50, Dezember 1991.

[29] Euler, Lutz. Die Programmierschnittstelle des ASL-Attributterm-Formalismus. Technischer Bericht ASL-Memo-48-92/UHH, Universität Hamburg, Bodenstedtstraße 16, D-W-2000 Hamburg 50, April 1992.

[30] Euler, Lutz. Syntax des ASL-Attributterm-Formalismus. Technischer Bericht ASL-Memo-45-92/UHH, Universität Hamburg, Bodenstedtstraße 16, D-W-2000 Hamburg 50, April 1992.

[31] Forney, G. D. „The Viterbi Algorithm“. *Proceedings of IEEE*, (1973), 61:266–278.

[32] Fujisaki, T. „An approach to stochastic parsing“. *Proceedings of COLING*, 1984.

[33] Fujisaki, T., F. Jelinek, J. Cocke, E. Black und T. Nishino. „A Probabilistic Parsing Method for Sentence Disambiguation“. Tomita, Masura, Hg., *Current Issues in Parsing Technology*, Norvell, Mass., 1991. Kluver Akademic Publishers. 139–148.

[34] Gazdar, G., E. Klein, G. K. Pullum und I. A. Sag. *Generalized Phrase Structure Grammar*. Cambridge, Mass.: Blackwell, 1985.

[35] Gerdemann, Dale. „Using Restriction to Optimize Unification Parsing“. *International Parsing Workshop*, 1989.

[36] Godden, K. „Improving the Efficiency of Graph Unification“. *Proc. of the ACL*, 1990.

[37] Görz, Günther. *Strukturanalyse natürlicher Sprache*. Bonn: Addison Wesley, 1988.

[38] Görz, Günther und Hans Weber. „Incremental Processing in ASL“. Weisweber, W., C. Hauenschild, S. Preus, B. Schmitz und C. Umbach, Hg., *Workshop on Text Understanding and Domain Modelling, Oktober 1991*, TU Berlin, Fachbereich Informatik, KIT, 1991.

[39] Hauenstein, A. und H. Weber. „An Investigation of Tightly Coupled Time Synchronous Speech Language Interfaces Using a Unification Grammar.“. McKevitt, Paul, Hg., *Proceedings of the Workshop on Integration of Natural Language and Speech Processing at AAAI 94*, Seattle, August 1994. 42–49.

[40] Hauenstein, A. und H. Weber. „An Investigation of Tightly Coupled Time Synchronous Speech Language Interfaces.“. *Proceedings of the CONVENTS 94*, Wien, September 1994.

[41] Hemphill, Charles und Joseph Picone. „Chart Parsing of Stochastic Spoken Language Models". *DARPA Workshop on Speech and Natural Language*, Philadelphia, Pensylvania, February 1989. DARPA.

[42] Herbrand, J. „Recherches sur la theorie de la demonstration.". Goldfarb, W., Hg., *Logical Writungs*. Harvard University Press, 1971.

[43] Huet, G. *Resolutions d'Equations dans les Langages d'Ordre 1,2,...* Diss. Universite de Paris VII, 1976.

[44] Jelinek, F. „Basic Methods of Probabilistic Context Free Grammars". Laface, P. und R. De Mori, Hg., *Speech Recognition and Understanding: Recent Advances*, Band NATO ASI Series, F 75, Berlin Heidelberg, 1992. Springer Verlag. 345–360.

[45] Jelinek, F., R. L. Mercer und L. R. Bahl. „A maximum likelihood approach to continuous speech recognition". *IEEE Tansactions on Pattern Analysis and Machine Intelligence*, 1983. 179–190.

[46] Jelinek, F. et. al. „The Developement of an Experimental Discrete Dictation Recognizer". *Proceedings of the IEEE*, Band 73/11, November 1985. 1616–1624.

[47] Johnson, Marc. *Attribute-Value Logic and the Thoery of Grammar*. Diss. Stanford University, 1987.

[48] Jurafsky, Daniel, Chuck Wooters, Gary Tajchman, Jonathan Segal, Andreas Stolcke und Nelson Morgan. „Integrating Experimental Models of Syntax, Phonology, and Accent/Dialoct in a Speech Recognizer.". McKevitt, Paul, Hg., *Proceedings of the Workshop on Integration of Natural Language and Speech Processing at AAAI 94*, Seattle, August 1994. 107–115.

[49] Katz, S. M. „Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer". *Proceedings of the IEEE ICASSP*, Band 35:3, March 1987. 400–401.

[50] Kay, Martin. „Functional Grammar". al, C. Chiarelloet, Hg., *Proceedings 5th Annual Meeting of the Berkeley Linguistics Society*, 1979.

[51] Kay, Martin. „Functional Unification Grammar". *Proceedings of the COLING*, 1984. 75–78.

[52] Kesselheim, M. und W. Bloemberg. „A system for creating and manipulating generalized wordclass transition matrices from large labeled text corpora.". *Proceedings of the COLING*, 1988. 49–53.

[53] Kita, K., T. Kawabata und H. Saito. „HMM Continuous Speech Recognition Using Predictive LR Parsing". *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1989. 703–706.

[54] Kneser, Reinhard und Hermann Ney. „Forming Word Classes by Statistical Clustering for Statistical Language Modelling.". *First Quantitative Linguistics Conference QUALICO*, University of Trier, Germany, September 1991.

[55] Kogure, Kiyoshi. „Parsing Japanese Spoken Sentences Based on HPSG“. *International Parsing Workshop*, 1989. 132–141.

[56] Lang, Bernard. „Deterministic Techniques For Efficient Non-deterministic Parsers.“. Loeckx, J., Hg., *Proc. of the 2nd Colloquium on Automata, Languages and Programming.*, Band 14 von *Lecture Notes in Compute Science*, Saarbrücken, 1974. Springer.

[57] Lee, Kai-Fu. *The Development of the SPHINX System.* Kluwer, 1989.

[58] Magermann, D.M. und M.P. Marcus. „Pearl: A probabilistic chart parser.“. *Proc. of the European ACL*, March 1991.

[59] Magermann, David M. *Natural Language Parsing as Statistical Pattern Recognition.* Diss. Stanford University, February 1994.

[60] Markov, A. A. „An example of statistical investigation in the text of *Eugene Onyegin* illustrating coupling of tests in chains“. *Proc. Acad. Sci. St. Petersburg*, (153-162 1913), VI(7):153–162.

[61] Moshier, M.D. und W.C. Rounds. „A Logic for Partially Specified Data Structures“. *Orc. of the 14th ACM Symposium on Principles of Programming Languages*, 1987. 156–167.

[62] Murveit, Hy und Robert Moore. „Integrating Natural Language Constraints into HMM-based Speech Recognition“. *IEEE International Conference on Acoustics, Speech and Signal Processing*, Band 1, Albuquerque, April 1990. 573–576.

[63] Nádas, A. „On Turings formula for word probabilities.“. *IEEE ICASSP*, (Dezember 1985), 33(6).

[64] Ney, H., R. Haeb-Umbach et al. „Improvements in Beam Search for 10000-Word Continuous Speech Recognition.“. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1992.

[65] Ney, Hermann. „Dynamic Programming Parsing for Context-Free Grammars in Continuous Speech Recognition“. *IEEE Transactions on Signal Processing*, (February 1991), 39(2):336–340.

[66] Ney, Hermann, „Architecture and Search Strategies for Large-Vocabulary Continuous-Speech Recognition.“. NAT)-ASI Bubión, 1993. 59–84.

[67] Paeseler, Annedore. „Modification of Earleys Algorithm for Speech Recognition“. *NATO ASI Series: Recent Advances in Speech Understanding*, (1988), F46:465–472.

[68] Pereira, F C.N. und David H.D. Warren. „Parsing as Deduction“. *Proc. 21th Annual Meeting of the Association for Computational Linguistics*, Cambridge, Massachusetts, 1983. 137–144.

[69] Pereira, F.C.N. „Grammars and Logic of Partial Information.“. *Proc. of the 4th International Conference on Logic Programming*, 1987. 989–1031.

[70] Pereira, F.C.N. und David H. D. Warren. „Definite Clause Grammars for Language Analysis – A Survey of the Formalism and a Comparison with Augmented Transition Networks“. *Artificial Intelligence*, (1980), 13:231–278.

[71] Pohl, I. „First Results on the Effect of Error in Heuristic Functions“. Meltzer, B. und D. Michie, Hg., *Machine Intelligence*. Band 5. New York: Elsevier Science Publ. Co., 1970. 219–236.

[72] Pollard, C. und I. A. Sag. *Information Based Syntax and Semantics*. Band 13 von *Lecture Notes*. Standford: CSLI, 1987.

[73] Poritz, Alan B. „Hidden Markov modells: A guided tour.“. *Proc. ICASSP*, (1988), 7–13.

[74] Rabiner, L. R. „Mathematical Foundations of Hidden Markov Models“. Niemann, H., M. Lang und G. Sagerer, Hg., *Recent Advances in Speech Underständing and Dialog Systems*, Heidelberg, 1988. Springer. 183–205.

[75] Rabiner, L. R. und B. H. Juang. „An Introduction to Hidden Markov Models“. *IEEE ASSP Magazine*, (January 1986), 4–16.

[76] Robinson, J. A. „A Machine-Oriented Logic Based on the Resolution Princple.“. *J. ACM*, (1965), 12(1).

[77] Rounds, W.C. und R.T. Kasper. „A Complete Logical Calculus of Record Structures Representing Linguistic Information“. *Proc. of the First IEEE Symposium on Logic in Computer Science.*, Boston, 1986. 38–43.

[78] Schabes, Y. „Polinomial time and space shift-reduce parsing of arbitrary context-free grammars.“. *Proc. of the ACL*, 1991. 106–113.

[79] Schmid, Ludwig. „Parsing word graphs using a linguistic grammar and a statistical language model.“. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1994. II 41 – II 44.

[80] Schwartz, Richard und Steve Austin. „Efficient, High-Performance Algorithms for N-Best Search“. *DARPA Speech and Natural Language Workshop*, Hidden Valley, Penn., June 1990. 6–11.

[81] Seneff, S. „TINA: A Probabilistic Syntactic Parser for Speech Understanding Systems.“. *DARPA Speech and Natural Language Workshop*, Philadephia, February 1989.

[82] Shannon, C. C. „A mathematical theory of communication“. *Bell Sys. Tech. J.*, (1948), 27:379–423, 623–656.

[83] Shannon, C. C. „Prediction and entropy of printed english“. *Bell Sys. Tech. J.*, (1951), 30:50–64.

[84] Shapiro, Stewart. *Encyclopedia of Artificial Intellingence*. 2nd Aufl. Wiley-Interscience Publication, 1992.

[85] Sheil, B.A. „Observations on contextfree parsing“. *Statistical Methods in Linguistics*, (1976), 6:71–109.

[86] Shieber, Stuart M. „Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms.“. *Proc. of the ACL 1985*, Band 23, 1985. 145–152.

[87] Shieber, Stuart M. *An Introduction to Unification-based Approaches to Grammar.* Lecture Notes. CSLI, 1986.

[88] Smolka, Gerd. A Feature Logic with Subsorts. Technischer Bericht LILOG Report Nr. 33, IBM IWBS, Stuttgart, Mai 1988.

[89] Staab, Steffen. GLR-Parsing von Worthypothesengraphen. Studienarbeit, Universität Erlangen, IMMD8, Dezember 1994.

[90] Thompson, H. „Best-first enumeration of paths through a lattice – an active chart parsing solution“. *Computer Speech and Language*, (1990), 4:263–274.

[91] Tomita, M. *An Efficient Context-free Parsing Algorithm for Natural Languages and Its Applications.* Diss. Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, May 1985.

[92] Tomita, M. „An Efficient Word Lattice Parsing Algorithm for Continous Speech Recognition.“. *ICASSP, Tokyo*, (1986), 1569–1572.

[93] Tomita, M. „An efficient augmented context-free parsing algorithm“. *Computational linguistics*, (1987), 13(1-2):31–46. GLR parsing with a unification grammar.

[94] Uszkoreit, Hans. „Categorial Unification Grammar“. *Proceedings of the 11th international Conference on Computational Linguistics, Bonn*, (1986), 86–100.

[95] Weber, Hans. Chartparsing in ASL-Nord: Berichte zu den Arbeitspaketen P1 bis P9. Technischer Bericht ASL-TR-28-92, Universität Erlangen-Nürnberg, IMMD8, 8525 Erlangen–Tennenlohe, Am Weichselgarten 9, Dezember 1992.

[96] Wirén, M. *Studies in Incremental Natural-Language Analysis.* Number Dissertation No. 292 in Linköping Studies in Science and Technology. Linköping University, 1992.

[97] Woods, W. A. „Optimal Search Strategies for Speech Understanding Control“. *Artificial Intelligence*, (1982), 18:295–326.

[98] Wright, J. H. „LR parsing of probabilistic grammars with input uncertainty for speech recognition.“. *Computer Speech and Language*, (1990), 4:297–323.

[99] Wrigley, E. N. und J. H. Wright. „Computational Requirements of Probabilistic LR Parsing in Speech Recognition using a Natural Language Grammar.“. *Proceedings of the Eurospeech in Genova*, (September 24.-26. 1991), 2:761–764.

[100] Younger, D.H. „Recognition and parsing of context-free languages in n^3“. *Information and Control*, (1967), 10(2):189–208.

[101] Zeevat, H., E. Klein und J. Calder. „Unification Categorial Grammar". Haddock, Klein und Morril, Hg., *Categorial Grammar, Unification Grammar and Parsing.* Edinburgh: Centre for Cognitive Science, University of Edinburgh., 1987.

Index

Anhang A

Grammatik

Im folgenden sind die Regeln der Grammatik angegeben, die in Kapitel 5 verwendet wurde. In Abschnitt 4 wurde eine Teilgrammatik der hier angegebenen verwendet.

A.1 Typhierarchie

Die Definition der Typhierarchie besteht aus is-a Deklarationen, die hier angegeben sind. Das Top-Element des Verbandes ist *t*. In jeder Deklaration ist der linkeste Typ Obertyp de folgenden Typen.

```
(define-type-module top nil *t*
  (isa *t* rule)
  (isa *t* sign)
  (isa sign lsign psign)
  (isa psign utterance s scomp srel v vorfeld mittelfeld nachfeld
       gruss bitte reladv np pp argcat adjcat time parg intro)
  (isa lsign p n pn comp vinf vfin prt modprt num strange det conj n-conj
       u-conj ppron pol gruss nummod relpron adv honprt uhr adject sil)
  (isa utterance wfr sfr a-e)
  (isa vorfeld np pp adv)
  (isa adv datum)
  (isa np ppron qpron)
  (isa mittelfeld arg adj)
  (isa nachfeld scomp srel)
  (isa v vinf vfin vp)
  (isa argcat np pp prt)
  (isa adjcat pp adv honprt negprt)
  (isa s s2 s1 sx)
  (isa time datum uhrzeit tageszeit)
  (isa parg np pn time)
  (isa conj n-conj u-conj)
  (isa intro gruss satzprt)
  (atom *t* yes no pos neg temp local from to start modal item kind-of
```

```
sing plur nom dat akk gen fem masc neut weak strong mixed auss-excl
wortfrage satzfrage)

  ;; goal



  ;; semantic types and ontology

  (isa *t* semsign ontology)
  (isa semsign connective quantifier)
  (isa ontology temporal abstract physical activity modality prop adjrel)
  (isa physical moveable non-moveable)
  (isa moveable animated non-animated selfmoving non-selfmoving)
  (isa non-moveable isa-place institution)
  (isa isa-place city building)
  (isa selfmoving vehicle human)
  (isa vehicle train)
  (isa prop connection day)
  (isa day weekday)
  (isa adjrel loc-adj dir-adj temp-adj spatial-adj modal-adj)
  (atom human speaker addressee)
  (atom city aachen aschaffenburg augsburg baden-baden bebra berlin bonn bremen
        bremerhaven dortmund duesseldorf emmerich etterzhausen frankfurt
        hamburg hannover heidelberg hindelang kiel koblenz koeln mannheim
        muenchen muenster nuernberg oldenburg osnabrueck paris regensburg rom
        saarbruecken stuttgart ulm wuerzburg)
  (atom connective and)
  (atom quantifier exists iota for-all several q-lambda)
  (atom activity fahren be suchen abfahren abgehen ankommen)
  (atom modality moechte muessen benoetigen brauchen pres etwa gern)
  (atom dir-adj source path goal)
  (atom adjrel dir-ab temp-ab loc-in zwischen heute temp-rel
        temp-um temp-an dir-auf bei temp-vor temp-gegen mit ohne)
  (atom prop object measure clocktime stunde halb set-of naechst
        ordinal woche abfahrtszeit angabe ankunft ankunftszeit aufenthalt
        auskunft bahngleis bahnhof bahnsteig bahnticket fahrkarte fahrt
        fahrtmoeglichkeit gleis grenzbahnhof ic-fahrt information klasse
        kurzaufenthalt lauf liegewagen moeglichkeit ostsee richtung schlafwagen
        schlafwagenabteil tagesausflug ticket umsteigen vorschlag
        zugfahrtmoeglichkeit zwischenstop ander billigst einzig ganz guenstig
        guenstigst letzt preiswert spaeter stuendlich uebernaechst)
  (atom  connection verbindung zugverbindung ic-verbindung bahnverbindung)
  (atom train zug eurocity ic intercity nachtzug)
  (atom day feiertag tag wochenende wochentag rosenmontag weihnachten
        heiligabend neujahr)
  (atom weekday montag dienstag mittwoch donnerstag freitag samstag sonntag
        sonnabend)
```

```
  (atom daytime abend nachmittag spaetabend morgen mittag vormittag nacht)
  )
```

A.2 Grammatikregeln

A.2.1 Startgraph

Der Startgraph wird beim Parsing in die Startkante eingefügt. Eine Analyse gilt als vollständig, wenn die Wurzel des Analysebaums mit dem Startgraphen unifizierbar ist.

```
utterance[(sem [(last nil)
                (quant [(last nil)])])]
```

A.2.2 Regeln mit kontextfreiem Typgerüst

Grammatikregeln sind Merkmalsstrukturen, die während Training und Analyse wie folgt interpretiert werden. Als Tochter des Merkmals *rule* beschreibt eine Liste die eigentliche Regel. Die Liste hat die Syntax < .. >. Die erste Merkmalsstrukture der Liste ist die linke Seite der Regel. Die folgenden Merkmalsstrukturen bilden die Rechte Seite einer Regel.

```
;;; Regel fuer den Startgraphen
;;; Die erste ist fuer Lattices mit Stille, die zweite fuer Saetze.

rule[(name [])
     (rule <
       utterance[(sem %1)]
       sil[]
       s[(sem %1=[])]
sil[]
       >)]

rule[(name [])
     (rule <
       utterance[(sem %1)
 (modus %2)
 (syn %3)
 ]
       s[(sem %1=[])
 (modus %2=[])
 (syn %3=[])
 ]
       >)]
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;; MODUS: WORTFRAGE

;;; Normal Matixsatz Vorfeld, finites Verb, Mittelfeld




rule[(name [])
     (rule <
    wfr[(sem [(content %12)
 (last %16)
 (quant [(list %10)
 (last %13)])])]
       sil[]
       %1=vorfeld[(syn [(wh yes)])
  (sem [(quant [(list %10=[])
(last %11=[])])])]
       vfin[(syn [(subcat <%1 . %2>)])
    (sem [(content %12=[(var %14=[])])
  (last %15)])]
       mittelfeld[(syn [(subcat %2=[])])
  (sem [(content [(var %14)
  (cond %15=[])])
(last %16=[])
(quant [(list %11)
(last %13=[])])])]
       sil[]
       >)]

;;; Normaler Matrixsatz Vorfeld, Finitum, Infinitivkonstruktion

rule[(name [])
     (rule <
    wfr[(sem [(content %12)
     (last %16)
     (quant [(list %10)
     (last %13)])])]
    sil[]
      %1=vorfeld[(syn [(wh yes)])
  (sem [(quant [(list %10=[])
       (last %11=[])])])]
      vfin[(syn [(subcat <%1 . %2>)])
   (sem [(content %12=[(var %14=[])])
```

```
 (last %15)])]
      vp[(syn [(subcat %2=[])] )
 (sem [(content [(var %14)
 (cond %15=[])])
       (last %16=[])
       (quant [(list %11)
    (last %13=[])])])])]
      sil[]
>)]

;;; Normaler Matrixsatz Vorfeld, Finitum.

rule[(name [])
     (rule <
    wfr[(sem %10=[(quant %11)])]
    sil[]
       %1=vorfeld[(syn [(wh yes)])
  (sem [(quant %11=[])])]
       vfin[(syn [(subcat <%1>)])
    (sem %10)]
       sil[]
       >)]

rule[(name [])
     (rule <
    wfr[(sem [(content [(cond [(conn and)
       (sub-wffs < %10 %11 >)])])
      (quant %12)])]
    sil[]
       %1=vorfeld[(syn [(wh yes)])
  (sem [(quant %12=[])])]
       vfin[(syn [(subcat <%1 . %2=[]>)])
    (sem %10=[(content %15=[(var %16=[])])
      (last %17)
      (quant %18)])]
       mittelfeld[(syn [(subcat %2)])
  (sem [(content [(var %16)
  (cond %17=[])])
(last nil)
(quant %18=[(last nil)])])]
       conj[(phon "UND")]
       vfin[(syn [(subcat <%1 . %3=[]>)])
    (sem %11=[(content [(var %20=[])])
      (last %21)
      (quant  %22)])]
       vp[(syn [(subcat %3)])
  (sem [(content [(var %20)
  (cond %21=[])])
```

```
(last nil)
(quant %22=[(last %13=[])])])]
sil[]
       >)]




;;; MODUS: AUSSAGE-EXCLAMATIV

;;; Normal Matixsatz Vorfeld, finites Verb, Mittelfeld



rule[(name [])
     (rule <
    a-e[(sem [(content %12)
      (last %16)
      (quant [(list %10)
      (last %13)])])]
       sil[]
       %1=vorfeld[(syn [(wh no)])
  (sem [(quant [(list %10=[])
(last %11=[])])])]
       vfin[(syn [(subcat <%1 . %2>)])
    (sem [(content %12=[(var %14=[])])
  (last %15)])]
       mittelfeld[(syn [(subcat %2=[])])
  (sem [(content [(var %14)
  (cond %15=[])])
(last %16=[])
(quant [(list %11)
(last %13=[])])])]
       sil[]
       >)]

;;; Normaler Matrixsatz Vorfeld, Finitum, Infinitivkonstruktion

rule[(name [])
     (rule <

    a-e[(sem [(content %12)
      (last %16)
      (quant [(list %10)
      (last %13)])])]
       sil[]
       %1=vorfeld[(syn [(wh no)])
```

```
  (sem [(quant [(list %10=[])
(last %11=[])])])]
       vfin[(syn [(subcat <%1 . %2>)])
    (sem [(content %12=[(var %14=[])])
  (last %15)])]
       vp[(syn [(subcat %2=[])] )
  (sem [(content [(var %14)
  (cond %15=[])])
(last %16=[])
(quant [(list %11)
(last %13=[])])])]
       sil[]
       >)]

;;; Normaler Matrixsatz Vorfeld, Finitum.

rule[(name [])
     (rule <
    a-e[(sem %10=[(quant %11)])]
       sil[]
       %1=vorfeld[(syn [(wh no)])
  (sem [(quant %11=[])])]
       vfin[(syn [(subcat <%1>)])
    (sem %10)]
       sil[]
       >)]


rule[(name [])
     (rule <

    a-e[(sem [(content [(cond [(conn and)
       (sub-wffs < %10 %11 >)])])
      (quant %12)])]
    sil[]
       %1=vorfeld[(syn [(wh no)])
  (sem [(quant %12=[])])]
       vfin[(syn [(subcat <%1 . %2=[]>)])
    (sem %10=[(content %15=[(var %16=[])])
      (last %17)
      (quant %18)])]
       mittelfeld[(syn [(subcat %2)])
  (sem [(content [(var %16)
  (cond %17=[])])
(last nil)
(quant %18=[(last nil)])])]
       conj[(phon "UND")]
       vfin[(syn [(subcat <%1 . %3=[]>)])
```

```
    (sem %11=[(content [(var %20=[])])
      (last %21)
      (quant  %22)])]
       vp[(syn [(subcat %3)])
  (sem [(content [(var %20)
  (cond %21=[])])
(last nil)
(quant %22=[(last %13=[])])])]
sil[]
       >)]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;; Normaler Fragesatz / Imperativ Finitum, Rest


rule[(name [])
     (rule <
       sfr[(sem [(content %13)
 (last %16)
 (quant %17)])]
    sil[]
       vfin[(syn[(subcat %2)])
    (sem [(content %13=[(var %11)])
  (last %12)])]
       mittelfeld[(syn [(subcat %2=[])])
  (sem [(content [(var %11=[])
  (cond %12=[])])
(last %16=[])
(quant %17=[])])]
       sil[]
       >)]



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

rule[(name [])
     (rule <
       s1[]
       vfin[]
       vinf[]
       >)]
```

```
;;;; Mittelfeldregeln


rule[(name [])
     (rule <
arg[(syn [(subcat <%1 . %2=[]>)])
    (adjdtrs %3=[])
    (sem [(content %12)
    (last %15)
    (quant [(list %10)
    (last %13)])])]
%1=argcat[(sem [(quant [(list %10=[])
    (last %11=[])])])]
mittelfeld[(syn [(subcat %2)])
   (adjdtrs %3)
   (sem [(content %12=[])
 (last %15=[])
 (quant [(list %11)
 (last %13=[])])])]
>)]

rule[(name [])
     (rule <
arg[(syn [(subcat <%1 . nil>)])
    (adjdtrs nil)
    (sem [
      (content [(cond %11=[])])
      (last %11)
      (quant %10=[])])]
%1=argcat[(sem [(quant %10)])]
>)]

rule[(name [])
     (rule <
adj[(syn [(subcat %2=[])])
    (adjdtrs <%3 . %1>)
    (sem [(content [(var %16)
    (cond < %12 . %17 >)])
  (last %13)
  (quant [(list %14)
  (last %15)])])]
%3=adjcat[(sem [(content [(cond %12=[])
  (var %16)])
(quant [(list %14=[])
(last %11)])])]
mittelfeld[(syn [(subcat %2)])
   (adjdtrs %1=[])
```

```
   (sem [(content [(var %16=[])
   (cond %17=[])])
 (last %13=[])
 (quant [(list %11=[])
 (last %15=[])])])]
>)]

rule[(name [])
     (rule <
adj[(syn [(subcat nil)])
    (adjdtrs <%1>)
    (sem [(content [(var %10)
    (cond < %11 . %12 >)])
  (last %12=[])
  (quant %13)])]
%1=adjcat[(sem [(content [(var %10=[])
  (cond %11=[])])
(quant %13=[])])]
>)]



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Regeln fuer die Infinitifkonstruktionen
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


rule[(name [])
     (rule <
vp[(syn[(subcat <%2>)])
   (sem [(content [(cond nil)])
 (quant %17)])]
mittelfeld[(syn [(subcat %1)])
   (sem [(content [(var %14)
   (cond %15=[])])
     (last nil)
     (quant %17=[])])]
%2=vinf[(syn [(subcat < [] . %1=[] >)])
(sem [(content [(var %14=[])])
 (last %15)])]
>)]




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Regeln fuer NPs
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; Standardregel

rule[(name [])
     (rule <
np[(syn [(wh %3)
 (agr %1)
 (case %2)])
   (sem %10)]
det[(syn [(wh %3=[])
  (agr %1)
  (case %2)])
    (sem %10)]
n[(syn [(agr %1=[])
(case %2=[])])
  (sem %10=[])]
>)]


rule[(name [])
     (rule <
    n[(xtype %2)
      (syn %1)
      (sem [(content %10)
    (last %12)])]
     adject[(xtype %2)
    (syn [(agr %3=[])])
    (sem [(content [(var  %13=[])
    (cond %11=[])])])]
     n[(xtype %2=[])
       (syn %1=[(agr %3)])
       (sem [(content %10=[(var %13)])
     (last < %11 . %12=[]>)])]
     >)]


; Regel fuer Personalpronomina (ueber Typhierarchie direkt als vorfeld)
;
; rule[(name [])
;     (rule <
;       np[(syn [(agr %1=[])
;                (case %2=[])])
;           (sem %10)]
;       ppron[(syn [(agr %1)
;                   (case %2)])
;              (sem %10=[])]
;       >)]
```

```
; Regel fuer NPs und (Standard-) Relativsaetze

rule[(name [])
     (rule <
np[(syn [(wh %4)
 (agr %1)
 (case %2)
 (rel yes)])]
  np[(syn [(wh %4=[])
   (agr %1)
   (case %2)
   (rel no)])]
  srel[(syn [(agr %1=[])
     (case %2=[])])]
>)]




rule[(name [])
     (rule <
   np[(syn %1=[])
      (sem [(content %13)
    (last %20=[])])]
np[(syn %1)
   (sem [(content %13=[(var %10)])
 (last < [(content %11)
   (quant %12)] . %20 >)])]
  pp[(sem [(content [(var %10=[])
     (cond %11=[])])
   (quant %12=[(last nil)])])]
>)]

rule[(name [])
     (rule <
np[(syn [(agr %1=[(num plur)])
 (case %2=[])
 (mass yes)])
   (sem %10)]
n[(syn [(agr %1)
(case %2)
(mass yes)])
  (sem %10=[])]
>)]

;;; Zuege Muenchen Nuernberg
```

```
rule[(name [])
     (rule <
    n[(xtype %2=local)
      (syn [(agr %1=[])
    (case %3=[])])
      (sem [(content %10)
    (last %20=[])])]
     n[(xtype %2)
       (syn [(agr %1)
     (case %3)])
       (sem [(content %10=[(var %11=[])])
     (last < [(content [(conn and)
(sub-wffs < [(pred source)
     (theme %11)
     (goal %14)]
  [(pred goal)
   (theme %11)
   (goal %15)] >)])
      (quant [(list < %13 %17 > )])] . %20
   >)])]
     pn[(sem [(content %13=[(var %14=[])])
      (last nil)])]
     pn[(sem [(content %17=[(var %15=[])])
      (last nil)])]
>)]
```

```
; Regeln fuer PPs
```

```
rule[(name [])
     (rule <
    pp[
   (syn [(subcat nil)
 (wh %3=[])
 (ptype %2)])
   (sem %10)]
      p[
(syn [(subcat <%1>)
      (ptype %2=[])])
(sem %10=[])]
      %1=parg[(syn [(wh %3)])]
      >)]
```

```
rule[(name [])
     (rule <
    pp[(ptype %2)
       (syn [(subcat nil)])
       (sem [(content [(var %10=[])
       (cond [(conn and)
      (sub-wffs < %11 %12 >)])])
     (quant %13)])]
      modprt[(sem [(content [(var %10)
     (cond %11=[])])])]
      pp[(ptype %2=[])
 (syn [(subcat nil)])
 (sem [(content [(var %10)
 (cond %12=[])])
       (quant %13=[])])]
      >)]
```

```
;;; Regel fuer Coordination

Normale Und-Regel

rule[(name [])
     (rule <
%1=[]
conj[(syn [(subcat <%1 %2>)])]
%2=[]
>)]
```

```
; Extra-Regel fuer zwischen

rule[(name [])
     (rule <pp[(syn [(ptype %4=[])
     (subcat nil)])
       (sem %10)]
      p[(phon "ZWISCHEN")
(syn [(ptype %4)
      (subcat <%1 %2 %3>)])
(sem %10=[])]
      %1=time[]
      %2=conj[]
```

```
      %3=time[]
      >)]



;;; Uhrzeiten


rule[(name [])
     (rule <
uhrzeit[(xtype temp)
   (sem [(content [(var %10=[])
   (cond [(pred clocktime)
  (inst %10)
  (hour %11)])])])]
num[(sem [(content [(cond [(arg2 %11=[])])])])]
>)]

rule[(name [])
     (rule <
uhrzeit[(xtype temp)
   (sem [(content [(var %10=[])
   (cond [(pred clocktime)
  (inst %10)
  (hour %11)])])])]
num[(sem [(content [(cond [(arg2 %11=[])])])])]
uhr[]
>)]

rule[(name [])
     (rule <
   uhrzeit[(xtype temp)
   (sem [(content [(var %10=[])
    (cond [(pred clocktime)
   (inst %10)
   (hour %11)
   (min %12)])])])]
num[(sem [(content [(cond [(arg2 %11=[])])])])]
uhr[]
num[(sem [(content [(cond [(arg2 %12=[])])])])]
>)]


rule[(name [])
     (rule <
   uhrzeit[(xtype temp)
   (sem [(content [(var %10)
   (cond [(conn and)
```

```
  (sub-wffs < %11 %12 >)])])])]
    uhrzeit[(sem [(content [(var %10=[])
    (cond %11=[])])
  (last nil)])]
    tageszeit[(sem [(quant [(list < [(var %10)
     (cond %12=[])] . %13=[]>)])])]
>)]

;;;;;; In die Typhierarchie eingefuegt , hw.;;;;
;rule[(name [])
;     (rule <
;           adv[(sem %10)]
;              datum[(sem %10=[])]
;              >)]
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;; Regel fuer Datumsangaben


rule[(name [])
     (rule <
    datum[]
 n[(xtype temp)
   (syn %1)]
 adject[(xtype %2)
(syn [(agr %3=[])])]
 n[(xtype %2=[])
   (syn %1=[(agr %3)])]
 >)]



rule[(name [])
     (rule <
    datum[(sem [(content [(var %15=[])
   (cond [(pred loc-in)
  (theme %15)
  (goal %10)])])
 (quant [(list < %12 . %16 >)
 (last %16=[])])])]
    adject[(xtype %2)
   (syn %3=[])
(sem [(content [(var %10)
(cond %11=[])])])]
    n[(xtype %2=temp)
      (syn %3)
   (sem [(content %12=[(var %10=[])])
```

```
     (last < %11 >)])]
 >)]
;;;;;;;;;;;;;;;;;;;;;

;Grussregel

rule[(name [])
     (rule<
   %1=utterance[]
       gruss[(syn [(subcat nil)])]
       %1
    >)]


rule[(name [])
     (rule<
   %1=utterance[]
       pol[(syn [(subcat nil)])]
       %1
    >)]

rule[(name [])
     (rule<
   utterance[(sem %10)]
    honprt[(syn [(subcat nil)])]
    vorfeld[(sem %10=[])]
    >)]



rule[(name [])
     (rule<
   gruss[(syn [(subcat nil)])]
   gruss[(syn[(subcat <%2>)])]
   %2=gruss[]
    >)]




;;; Etwa, ungefaehr...

rule[(name [])
     (rule<
      time[(modified yes)
   (sem %10)]
      nummod[(sem %10=[(content [(var %12)
 (cond [(theme %11)])])])]
```

```
   time[(modified no)
(sem [(content [(var %12=[])
(cond %11=[])])])]
       >)]



rule[(name [])
     (rule<
   pp[(modified yes)
      (ptype temp)
   (sem %10)]
   nummod[(sem %10=[(content [(var %12)
      (cond [(theme %11)])])
    (quant %13)])]
   pp[(modified no)
      (ptype temp)
      (sem [(content [(var %12=[])
      (cond %11=[])])
    (quant %13=[])])]
       >)]
```

A.3 Ausgewählte Lexikoneinträge

Das gesamte Lexikon würde ausgedruckt etwa 120 Seiten ergeben. Daher sind hier nur ausgewählte Einträge aufgeführt.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                         ;;;
;;;                Lexikon fuEr das Zugauskunftssystem                      ;;;
;;;                                                                         ;;;
;;;                                                                         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                         ;;;
;;;                               Nomina                                    ;;;
;;;                                                                         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Stille als Papierkorbkategorie

"-sil"

sil[]

"abends"
```

```
n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (gen fem)
      (num sing)])
(case gen)])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred abend)
      (inst %10)] . %11>)])])
(last %11=[])])]


"abfahrtszeit"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (gen fem)
      (num sing)])
(case {nom, akk})])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred abfahrtszeit)
      (inst %10)] . %11>)])])
(last %11=[])])]


"bahnticket"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (gen neut)
      (num sing)])
(case {nom, dat, akk})])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred bahnticket)
      (inst %10)] . %11>)])])
(last %11=[])])]


"bahnverbindung"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
```

```
      (gen fem)
      (num sing)])])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred bahnverbindung)
      (inst %10)] . %11>)])])
(last %11=[])])]

"eurocity"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (gen masc)
      (num sing)])
(case {nom, dat, akk})])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred eurocity)
      (inst %10)] . %11>)])])
(last %11=[])])]

"moEglichkeit"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (gen fem)
      (num sing)])])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred moeglichkeit)
      (inst %10)] . %11>)])])
(last %11=[])])]

"moEglichkeiten"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (num plur)])])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred moeglichkeit)
      (inst %10)] . %11>)])])
(last %11=[])])]

"nacht"
```

```
n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (gen fem)
      (num sing)])])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred nacht)
      (inst %10)] . %11>)])])
(last %11=[])])]

"zugverbindungen"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (num plur)])])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred zugverbindung)
      (inst %10)] . %11 >)])])
(last %11=[])])]

"zwischenstop"

n[(syn [(wh no)
(mass no)
(agr [(pers 3)
      (gen masc)
      (num sing)])
(case {nom, dat, akk})])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred zwischenstop)
      (inst %10)] . %11 >)])])
(last %11=[])])]



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                         ;;;
;;;                           Staedtenamen                                  ;;;
;;;                                                                         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


"aachen"
```

```
pn[(syn [(xtype local)
 (wh no)
 (agr [(pers 3)
       (gen neut)
       (num sing)])])
   (sem [(content [(var %1= [])
   (cond [(conn and)
  (sub-wffs < [(pred aachen)
       (inst %1)] . %10=[] > )])])
 (last %10)])]

"aschaffenburg"

pn[

   (syn [(xtype local)
 (wh no)
 (agr [(pers 3)
       (gen neut)
       (num sing)])])
   (sem [(content [(var %1= [])
   (cond [(conn and)
  (sub-wffs < [(pred aschaffenburg)
       (inst %1)] . %10=[] > )])])
 (last %10)])]


"augsburg"

pn[
   (syn [(xtype local)
 (wh no)
 (agr [(pers 3)
       (gen neut)
       (num sing)])])
   (sem [(content [(var %1= [])
   (cond [(conn and)
  (sub-wffs < [(pred augsburg)
       (inst %1)] . %10=[] > )])])
 (last %10)])]

"wuErzburg"

pn[
   (syn [(xtype local)
 (wh no)
 (agr [(pers 3)
```

```
       (gen neut)
       (num sing)])])
   (sem [(content [(var %1= [])
   (cond [(conn and)
  (sub-wffs < [(pred wuerzburg)
       (inst %1)] . %10=[] > )])])
 (last %10)])]



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                       Wochentage, Monate, ...                          ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; siehe auch Wochentage als "datum[]"


"samstag"

n[(syn [(agr [(pers 3)
      (gen masc)
      (num plur)])])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred samstag)
      (inst %10)] . %11 >)])])
(last %11=[])])]


"rosenmontag"

n[(syn [(agr [(pers 3)
      (gen masc)
      (num plur)])])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred rosenmontag)
      (inst %10)] . %11 >)])])
(last %11=[])])]

"weihnachten"

n[(syn [(agr [(pers 3)
      (gen neut)
      (num sing)])])
```

```
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred weihnachten)
      (inst %10)] . %11 >)])])
(last %11=[])])]

"mittag"

n[(syn [(agr [(pers 3)
      (gen masc)
      (num sing)])
(case {nom, dat, akk})])
  (sem [(content [(var %10=[])
  (cond [(conn and)
 (sub-wffs < [(pred mittag)
      (inst %10)] . %11 >)])])
(last %11=[])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                           Determiner                                   ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"alle "

det[(syn [(wh no)
  (agr [(pers 3)
(type strong)
(num plur)])
  (case {nom, akk})])
    (sem [(content [(qforce for-all)])])]

"aller"

det[(syn [(wh no)
  (agr [(pers 3)
(type strong)
(num plur)])
  (case gen)])
    (sem [(content [(qforce for-all)])])]

"das"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen neut)
```

```
(num sing)])
  (case {nom, akk})])
    (sem [(content [(qforce iota)])])]

"dem "

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen masc)
(num sing)])
  (case dat)])
    (sem [(content [(qforce iota)])])]

"den"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen masc)
(num sing)])
  (case akk)])
    (sem [(content [(qforce iota)])])]

"den"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(num plur)])
  (case dat)])
    (sem [(content [(qforce iota)])])]

"der "

det[(syn [(wh no)
  (agr [(pers 3)
(gen masc)
(num sing)])
  (case nom)])
    (sem [(content [(qforce iota)])])]

"der "

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen fem)
```

```
(num sing)])
  (case dat)])
    (sem [(content [(qforce iota)])])]

"der"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(num plur)])
  (case gen)])
    (sem [(content [(qforce iota)])])]

"die"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen fem)
(num sing)])
  (case {nom, akk})])
    (sem [(content [(qforce iota)])])]

"die"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(num sing)])
  (case {nom, akk})])
    (sem [(content [(qforce iota)])])]

"diese"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen fem)
(num sing)])
  (case nom)])
    (sem [(content [(qforce iota)])])]

"diese"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(num plur)])
```

```
  (case nom)])
    (sem [(content [(qforce iota)])])]

"dieser"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen masc)
(num sing)])
  (case nom)])
    (sem [(content [(qforce iota)])])]

"dieses"

det[(syn [(wh no)
  (agr [(pers 3)
(type weak)
(gen neut)
(num sing)])
  (case nom)])
    (sem [(content [(qforce iota)])])]

"ein"

det[(syn [(wh no)
  (agr [(pers 3)
(gen masc)
(type mixed)
(num sing)])
  (case {nom,akk})])
    (sem [(content [(qforce exists)])])]


"ein"

det[(syn [(wh no)
  (agr [(pers 3)
(gen neut)
(type mixed)
(num sing)])
  (case {nom,akk})])
    (sem [(content [(qforce exists)])])]

"eine"

det[(syn [(wh no)
  (agr [(pers 3)
```

```
(gen fem)
(type mixed)
(num sing)])
  (case {nom, akk})])
    (sem [(content [(qforce exists)])])]

;;; einem Intercity, einem Wochentag, einem Zwischenstop

"einem"

det[(syn [(wh no)
  (agr [(pers 3)
(type mixed)
(gen masc)
(num sing)])
  (case dat)])
    (sem [(content [(qforce exists)])])]

"einen"

det[(syn [(wh no)
  (agr [(pers 3)
(type mixed)
(gen fem)
(num sing)])
  (case akk)])
    (sem [(content [(qforce exists)])])]


;;; in einer halben Stunde

"einer"

det[(syn [(wh no)
  (agr [(pers 3)
(type mixed)
(gen fem)
(num sing)])
  (case dat)])
    (sem [(content [(qforce exists)])])]

"keine"

det[(syn [(wh no)
  (agr [(pers 3)
(type mixed)
(num plur)])
  (case nom)])
```

```
    (sem [(content [(qforce no)])])]

"keiner"

det[(syn [(wh no)
  (agr [(pers 3)
(type mixed)
(gen masc)
(num sing)])
  (case no)])
    (sem [(content [(qforce no)])])]

"mehrere"

det[(syn [(wh no)
  (agr [(pers 3)
(type strong)
(num plur)])
  (case {nom, akk})])
    (sem [(content [(qforce several)])])]

"welche"

det[(syn [(wh yes)
  (agr [(pers 3)
(type weak)
(num plur)])
  (case {nom, akk})])
    (sem [(content [(qforce q-lambda)])])]

"welche"

det[
    (syn [(wh yes)
  (agr [(pers 3)
(type weak)
(gen fem)
(num sing)])
  (case {nom, akk})])
    (sem [(content [(qforce q-lambda)])])]

"welchem"

det[
    (syn [(wh yes)
  (agr [(pers 3)
(type weak)
(gen {masc, neut})
```

```
(num sing)])
  (case dat)])
    (sem [(content [(qforce q-lambda)])])]

"welchen"

det[
    (syn [(wh yes)
  (agr [(pers 3)
(gen masc)
(num sing)])
  (case akk)])
    (sem [(content [(qforce q-lambda)])])]

"welchen"

det[
    (syn [(wh yes)
  (agr [(pers 3)
(type weak)
(num plur)])
  (case dat)])
    (sem [(content [(qforce q-lambda)])])]

"welcher"

det[
    (syn [(wh yes)
  (agr [(pers 3)
(type weak)
(gen masc)
(num sing)])
  (case nom)])
    (sem [(content [(qforce q-lambda)])])]

"welches"

det[
    (syn [(wh yes)
  (agr [(pers 3)
(type weak)
(gen fem)
(num sing)])
  (case nom)])
    (sem [(content [(qforce q-lambda)])])]
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                          ;;;
;;;                             Adjektive                                    ;;;
;;;                                                                          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"andere"

adject[(syn [(agr [(type weak)
   (num sing)
   (gen fem)])
     (case nom)])
       (sem [(content [(var %10=[])
       (cond [(pred ander)
      (arg1 %10)])])])]


"andere"

adject[(syn [(agr [(type strong)
   (num plur)])
     (case {nom,akk})])
       (sem [(content [(var %10=[])
       (cond [(pred ander)
      (arg1 %10)])])])]

"letzten"

adject[(syn [(agr [(type weak)
   (num sing)
   (gen fem)])
     (case dat)])
       (sem [(var %10=[])
     (cond [(pred ordinal)
    (theme %10)
    (goal letzt)])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                          ;;;
;;;                              Zahlen                                      ;;;
;;;                                                                          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Kardinalzahlen

"sieben"

num[(sem [(content [(var %10=[])
```

```
    (cond [(pred measure)
   (arg1 %10)
   (arg2 7)])])])]

"dreiundzwanzig"

num[(sem [(content [(var %10=[])
    (cond [(pred measure)
   (arg1 %10)
   (arg2 23)])])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                      ;;;
;;;                        Personalpronomen                              ;;;
;;;                                                                      ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"sie"

ppron[(syn [(wh no)
    (agr [(pers 2)
  (num {sing,plur})])
    (case {nom, akk})
    (hon yes)])
      (sem [(content [(var %10=[])
      (cond [(conn and)
     (sub-wffs < [(pred object)
  (inst %10)] . %11>)])])
    (last %11=[])])]

"ich"

ppron[(syn [(wh no)
    (subcat nil)
    (case nom)
    (agr [(pers 1)
  (num sing)])])
      (sem [(content [(var %10=[])
      (cond [(conn and)
     (sub-wffs < [(pred speaker)
  (inst %10)] . %11>)])])
    (last %11=[])])]

"mir"

ppron[(syn [(wh no)
    (agr [(pers 1)
  (num sing)])
```

```
    (case dat)])
      (sem [(content [(var %10=[])
      (cond [(conn and)
     (sub-wffs < [(pred speaker)
  (inst %10)] . %11>)])])
    (last %11=[])])]
```

"er"

```
ppron[(syn [(wh no)
    (agr [(pers 3)
  (gen masc)
  (num sing)])
    (case nom)])
      (sem [(content [(var %10=[])
      (cond [(conn and)
     (sub-wffs < [(pred object)
  (inst %10)] . %11>)])])
    (last %11=[])])]
```

"es"

```
ppron[(syn [(wh no)
    (agr [(pers 3)
  (case neut)
  (num plur)])
    (case nom)])
      (sem [(content [(var %10=[])
      (cond [(conn and)
     (sub-wffs < [(pred object)
  (inst %10)] . %11>)])])
    (last %11=[])])]
```

"man"

```
ppron[(syn [(wh no)
    (agr [(pers 3)
  (gen masc)
  (num sing)])
    (case nom)])
      (sem [(content [(var %10=[])
      (cond [(conn and)
     (sub-wffs < [(pred object)
  (inst %10)] . %11>)])])
    (last %11=[])])]
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                          Prepositions                                  ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"ab"

p[(syn [(subcat <pn[(syn [(xtype local)
  (case dat)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])]>)
(ptype from)])
  (sem [(content [(var %3= [])
  (cond [(pred dir-ab)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"ab"

p[(syn [(ptype %1=temp)
(subcat <np[(syn [(xtype %1)
  (case dat)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred temp-ab)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

;;; "am" ist zusammengesetzt aus "an dem", wichtig fuEr Adjektivdeklination.
;;; die Nominalphrase darf keinen Artikel enthalten!!


"am"

p[(syn [(ptype %1=temp)
(subcat <n[(syn [(xtype %1)
 (case dat)])
   (sem [(content %4=[(var %2= [])
      (qforce iota)])
 (last nil)])]>)])
  (sem [(content [(var %3= [])
```

```
  (cond [(pred temp-an)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"an"

p[(syn [(ptype %1=temp)
(subcat <np[(syn [(xtype %1)
  (case dat)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred temp-an)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"auf"

p[(syn [(ptype local)
(subcat <np[(syn [(case dat)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred dir-auf)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"bei"

p[(syn [(subcat <np[(syn [(case dat)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred bei)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"im"

p[(syn [(subcat <n[(syn [(case dat)])
```

```
   (sem [(content %4=[(var %2= [])
      (qforce iota)])
 (last nil)])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred loc-in)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"in"

p[(syn [(ptype %1)
(subcat <parg[(syn [(case dat)
    (xtype %1=[])])
      (sem [(content %4=[(var %2=[])])
    (last nil)])]>)])
  (sem [(content [(var %10=[])
  (cond [(pred loc-in)
 (theme %10)
 (goal %2)])])
(quant [(list < %4 . %5 >)
(last %5=[])])])]

;; parg als Obertyp von
;; pn und np eigefuEhrt
;; Spart die Disjunktion !

"nach"

p[(syn [(ptype to)
(subcat < parg [(syn [(case akk)
      (xtype local)])
(sem [(content %4=[(var %2=[])])
      (last nil)])]>)])
  (sem [(content [(var %11=[(sorte [(dynamic yes)])])
  (cond [(pred goal)
 (theme %11)
 (goal %2)])])
(quant [(list < %4 . %5 >)
(last %5=[])])])]


"um"

p[(syn [(ptype temp)
(subcat <uhrzeit[(sem [(content %4=[(var %2= [])])
  (last nil)])]>)])
```

```
  (sem [(content [(var %3= [])
  (cond [(pred temp-um)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"von"

p[(syn [(ptype from)
(subcat <parg[(syn [(case dat)
    (xtype local)])
      (sem [(content %4=[(var %2=[])])
    (last nil)])]>)])
  (sem [(content [(var %1=[(sorte [(dynamic yes)])])
  (cond [(pred source)
 (theme %1)
 (goal %2)])])
(quant [(list <%4 . %5=[]>)
(last %5)])])]

"vor"

p[(syn [(ptype temp)
(subcat <parg[(syn [(case dat)])
      (sem [(content %4=[(var %2=[])])
    (last nil)])]>)])
  (sem [(content [(var %1=[])
  (cond [(pred temp-vor)
 (theme %1)
 (goal %2)])])
(quant [(list <%4 . %5=[]>)
(last %5)])])]

"zwischen"

p[(phon "ZWISCHEN")
  (syn [(ptype temp)
(subcat
;        {
;               <datum[] conj[] datum[]>,
 <uhrzeit[(sem [(content %12=[])])]
 conj[]
 uhrzeit[(sem [(content %13=[])])]>
;               }
 )])
  (sem [(content [(var %10=[])
  (cond [(pred zwischen)
```

```
 (theme %10)
 (goal %11)])])
(quant [(list < [(var %11=[])
 (cond [(pred set-of)
(inst %11)
(theme < %12  %13 >)])] . %20 >)
(last %20=[])])])]


>)])]

"uEber"

p[(syn [(subcat <parg[(case {dat, akk})
      (sem [(content %4=[(var %2=[])])
    (last nil)])]>)])
  (sem [(content [(var %1=[(sorte [(dynamic yes)])])
  (cond [(pred path)
 (theme %1)
 (goal %2)])])
(quant [(list < %4 . %5 >)
(last %5=[])])])]

;;;                       ===== wegen "uEber den Feiertagen"!


"gegen"

p[(syn [(subcat <np[(syn [(case akk)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred temp-gegen)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

"mit"

p[(syn [(subcat <np[(syn [(case dat)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred mit)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
```

```
(last %5=[])])])]


"ohne"

 p[(syn [(subcat <np[(syn [(case akk)])
    (sem [(content %4=[(var %2= [])])
  (last nil)])])]>)])
  (sem [(content [(var %3= [])
  (cond [(pred ohne)
 (theme %3)
 (goal %2)])])
(quant [(list < %4 . %5>)
(last %5=[])])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                             Modalpartikel                              ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


"mindestens "

modprt[(syn [(subcat <np[]>)])]

"mindestens "

modprt[(syn [(subcat <pp[]>)])]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                               Adverbien                                ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"gern"

adv[(sem [(content [(var %10=[])
    (cond [(pred gern)
   (theme %10)])])
  (quant [(list %11=[])
  (last %11)])])]

"gerne"

adv[(sem [(content [(var %10=[])
```

```
    (cond [(pred gern)
   (theme %10)])])
  (quant [(list %11=[])
  (last %11)])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                          ;;;
;;;                           polarity item                                  ;;;
;;;                                                                          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"gut"

pol[]

"prima"

pol[]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                          ;;;
;;;                                 Datum                                    ;;;
;;;                                                                          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


"heute"

datum[(syn [(xtype temp)
    (wh no)])
      (sem [(content [(var %10=[])
      (cond [(pred loc-in)
     (theme %10)
     (goal %11=[])])])
    (quant [(list <[(var %11)
    (cond [(pred heute)
   (inst %11)])] . %13>)
    (last %13=[])])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                          ;;;
;;;                            Tageszeit                                     ;;;
;;;                                                                          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"vormittags"

tageszeit[(syn [(wh no)])
```

```
  (sem [(content [(var %10=[])
  (cond [(pred temp-rel)
 (theme %10)
 (goal %11)])])
(quant [(list < [(var %11=[])
 (cond [(pred vormittag)
(inst %11)])] . %12>)
(last %12=[])])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                                Uhrzeit                                 ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"mittag"

uhrzeit[(syn [(xtype temp)
      (wh no)])
(sem [(content [(var %10=[])
(cond [(conn and)
       (sub-wffs < [(pred mittag)
    (inst %10)] . %20 >)])])
      (last %20=[])])]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                                Reste                                   ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; nummod

"zirka"

nummod[]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Superlative

"schnellsten"

adject[]

"besten"
```

```
adject[]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; abgetrennte Verbpraefixe

"weg"

prt[]

"los"

prt[]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; nebenordnende Konjunktion

"aber"

n-conj[]

"entweder"

n-conj[]

"oder"

n-conj[]

"dasS "

u-conj[]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; "bitte" und "danke"

"bitte"

honprt[(phon "bitte")]

"danke"

honprt[(phon "danke")]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;; GruEsse

"tag"

gruss[(phon "TAG")]

"morgen"

gruss[(phon "MORGEN")]

;;; "GruEsS Gott..."

"gott"

gruss[(phon "GOTT")]

;;; "Guten Tag"

"guten"

gruss[(phon "GUTEN")
      (syn [(subcat <gruss[(phon "MORGEN" )]>)])]

;;; "vielen Dank"

"vielen"

adject[]

"dank"

n[(syn [(agr [(pers 3)
      (gen masc)
      (num sing)])
(case nom)])]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Uhrzeiten


;; nur in Zeitangaben wie "acht Uhr"   ????

"uhr"

n[(syn [(agr [(pers 3)
      (gen fem)
      (num sing)])])]
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                              Verben                                    ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"abfahren"

vinf[(syn [(subcat <np[(sem [(content [(var %2=[])])])]>)])
     (sem [(content [(var %1=[])
     (cond [(conn and)
    (sub-wffs < [(pred abfahren)
(inst %1)
(agent %2)] . %3 >)])])
   (last %3=[])])]

"abgeht"

vfin[(syn [(subcat <np[(syn [(case nom)])
       (sem [(content %4= [(var %2=[])])
     (last nil)
     (quant [(list < %4 . %5 >)
     (last %5=[])])])])]>)])
     (sem [(content [(var %1=[])
     (cond [(conn and)
    (sub-wffs < [(pred abgehen)
(inst %1)
(agent %2)] . %3 >)])])
   (last %3=[])])]


"bitte" ;; Eine Fahrkarte bitte. Verbfreie Konstruktion

vfin[(syn [(subcat <np[(syn [(case akk)])]>)])]

"bleiben"

vinf[(syn [(subcat <np[]
      pp[(syn [(ptype local)])]>)])]


"brauche"

vfin[(syn [(subcat <np[(syn [(agr [(pers 1)
   (num sing)])
     (case nom)])
```

```
       (sem [(content %4= [(var %2)])
     (last nil)
     (quant [(list < %4 . %5 >)
     (last %5=[])])])])]
   np[(syn [(case akk)])
       (sem [(content %14= [(var %3)])
     (last nil)
     (quant [(list < %14 . %15 >)
     (last %15=[])])])])]>)])
     (sem [(content [(var %1= [(sorte [(dynamic no)])])
     (cond [(conn and)
    (sub-wffs < [(pred brauchen)
 (inst %1)
 (agent %2= [])
 (theme %3= [])]
       [(pred pres)
       (inst %1)] . %20> )])])
   (last %20=[])])]

"brauche"

vfin[(syn [(subcat <np[(syn [(case akk)])
       (sem [(content %14= [(var %3)])
     (last nil)
     (quant [(list < %14 . %15 >)
     (last %15=[])])])]
      np[(syn [(agr [(pers 1)
     (num sing)])
       (case nom)])
 (sem [(content %4= [(var %2)])
       (last nil)
       (quant [(list < %4 . %5 >)
       (last %5=[])])])])]>)])
     (sem [(content [(var %1= [(sorte [(dynamic no)])])
     (cond [(conn and)
    (sub-wffs < [(pred brauchen)
 (inst %1)
 (agent %2= [])
 (theme %3= [])]
[(pred pres)
 (inst %1)] . %20>)])])
   (last %20=[])])]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                        ;;;
;;;                                RESTE                                   ;;;
;;;                                                                        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
"etwa"

nummod[(sem [(content [(cond [(pred etwa)])])])]

"noch"

time[(syn [(subcat <time[]>)])]

"und"

conj[(phon "UND")
     (syn [(subcat
    {
;            <uhrzeit[] uhrzeit[]>
     <s[] s[]>,
     <np[] np[]>,
     <n[] n[]>,
;            <p[] p[]>,
     <pp[] pp[]>,
     <adv[] adv[]>
}
    )])]

"wenn's "

comp[(syn [(subcat <vfin[(syn [(agr [(num sing)
     (pers 3)])])]>)
   (wh yes)])]
```

Anhang B

Daten

Die im folgenden aufgeführten 203 Äußerungen wurden in dieser Arbeit verwendet. Aufgenommen und transkribiert wurden sie vom Phonetik Verbund PHONDAT[1]

Die Sätze 530 bis 539 waren in allen Experimenten die Testsätze. Die übrigen Sätze wurden als Lernstichprobe verwendet. Von der Lernstichprobe konnten 40 % nicht überspannend geparst werden. In diesen Fällen gingen die Fragmente in das Training ein.

```
499  Ich moEchte gerne nach Paris.
500  Kann man auch ohne Umsteigen fahren.
501  Guten Tag wann geht morgen vormittag ein Zug nach Frankfurt.
502  Wann geht der naEchste Zug nach Mannheim.
503  Kann ich am Samstag abend nach zweiundzwanzig Uhr noch von
     Frankfurt nach MuEnchen kommen.
504  Ich moEchte am ersten Feiertag nach Koblenz fahren.
505  Geht heute noch ein Zug nach Hannover.
506  Ich will am Montag um acht Uhr nach Stuttgart.
507  Darf ich eine IC-Fahrt zwischendurch unterbrechen.
508  Ich moEchte morgen abend nach KoEln fahren.
509  Wann geht heute der letzte Zug von Ulm nach MuEnchen.
510  Ich moEchte ab Ulm mit dem Zug nach Aschaffenburg fahren.
511  Kann ich heute noch mit einem Intercity nach Hamburg fahren.
512  Wann kann ich morgen fruEh nach MuEnchen fahren.
513  Ich musS uEber Hannover nach Hamburg fahren.
514  Wie kann ich von Ulm nach DuEsseldorf kommen.
515  Kann man heute noch von Ulm nach Koblenz kommen.
516  KoEnnten sie mir eine Verbindung nach Emmerich Grenzbahnhof geben.
517  Kann man direkt von Ulm nach Frankfurt fahren.
518  KoEnnen sie mir sagen wann ich spaEtestens in MuEnchen losfahren
     musS wenn ich noch vor zehn Uhr in Augsburg sein will.
519  Jetzt am Samstag nach Ulm wann fahren da morgens ZuEge.
520  Ich brauche den naEchsten Zug nach MuEnchen.
```

[1] PHONDAT wurde als Teil des Projektes ASL zwischen 1989 und 1992 vom BMFT gefördert, um Daten für die Forschung im Bereich Spracherkennung zur Verfügung zu stellen. PHONDAT gerhörten die Universität Kiel, Universität Bonn und LMU München an.

521 Ich will morgen abend nach Frankfurt.
522 Ich moEchte heute in vierzehn Tagen nach Rom fahren.
523 Wann kann ich heute noch nach Ulm fahren.
524 Wann geht morgen fruEh ein Zug nach Frankfurt.
525 Ich musS nach MuEnchen.
526 Guten Morgen ich moEchte heute nach neun von MuEnchen nach Hamburg.
527 Ich moEchte mit dem Intercity nach Hamburg fahren und zwar morgen.
528 Ich will mit dem Zug nach Hamburg und moEchte dort etwa gegen sieben Uhr ankommen.
529 Ich brauche fuEr uEbernaEchsten Montag nachmittag eine Zugverbindung von BadenBaden nach Oldenburg.
530 Ich moEchte von MuEnchen uEber NuErnberg nach Hamburg fahren.
531 Ich bin in KoEln und moEchte in einer halben Stunde nach MuEnchen fahren.
532 Ich brauche heute eine Verbindung ab KoEln.
533 Ich muEsSte uEber DuEsseldorf nach Hamburg fahren.
534 Ich benoEtige eine Zugverbindung von MuEnchen nach Aachen.
535 Guten Morgen ich moEchte heute zwischen acht und elf Uhr abends in Hamburg sein.
536 Ich suche ZuEge MuEnchen NuErnberg etwa um Mittag.
537 Ich brauche eine IC-Verbindung von Heidelberg nach Bebra naEchste Woche.
538 Ja ich moEchte gerne nach Hamburg fahren.
539 Bitte eine Verbindung von MuEnchen nach Frankfurt.
540 Guten Morgen ich moEchte heute nach BadenBaden von Ulm uEber WuErzburg fahren.
541 Ich moEchte in vierzehn Tagen von MuEnchen uEber Hannover nach Hamburg fahren.
542 Ich brauche Informationen uEber Zugverbindungen nach SaarbruEcken.
543 Wann faEhrt samstags der letzte Zug nach KoEln los.
544 Ich moEchte mit dem ersten Zug von MuEnchen nach NuErnberg fahren.
545 Ich moEchte heute mit Umsteigen in Bremen von MuEnchen nach Bremerhaven fahren.
546 Ich brauche diese Woche noch einen Intercity von MuEnchen nach Hamburg uEber NuErnberg.
547 Gibt es heute abend noch einen Zug nach WuErzburg.
548 Ich moEchte den Samstag vor Weihnachten nach Hamburg fahren und zwar uEber Nacht.
549 Morgen musS ich um sechs in Mannheim sein.
550 Wann ist der naEchste IC nach Mannheim.
551 Guten Tag ich braEuchte eine Verbindung nach Mannheim fuEr morgen.
552 Ich haEtte gerne eine Verbindung von WuErzburg nach NuErnberg fuEr Samstag moEglichst fruEh hin und sonntags moEglichst spaEt zuruEck.
553 Ich will einen Tagesausflug nach NuErnberg machen und entweder nachmittags gegen vier Uhr oder mit der letzten Verbindung abends zuruEckkommen.
554 Ich musS nach Dortmund aber mit einem Kurzaufenthalt in KoEln.
555 Gibt es einen Nachtzug der am dreiundzwanzigsten zwoElften morgens in Ulm ankommt.
556 Wie lange faEhrt man von hier nach OsnabruEck ungefaEhr.

557 Nennen Sie mir die guEnstigste Zugverbindung nach Kiel.
558 Wann und von welchem Bahnsteig fahren heute abend so ab sechzehn Uhr ZuEge nach Frankfurt ab.
559 Welches ist die guEnstigste Verbindung von Ulm nach Augsburg fuEr Sonntag moEglichst ab mittags hin.
560 Um spaEtestens um sechs Uhr in Hamburg zu sein wann musS ich da hier losfahren.
561 Welcher Zug faEhrt nicht vor neun Uhr nach Dortmund uEber KoEln.
562 Fahren keine ZuEge vor siebzehn Uhr von KoEln nach Dortmund.
563 Gibt es einen Zug der am dreiundzwanzigsten zwoElften um zirka zwanzig Uhr in Oldenburg ist.
564 Wenn ich jetzt losfahre wann waEre ich dann in Hamburg.
565 Ich haEtte gerne eine Auskunft uEber die Abfahrtszeit der ZuEge die ab zirka achtzehn Uhr von NuErnberg nach Augsburg fahren.
566 Ich moEchte an einem Wochentag nach Hamburg fahren.
567 GruEsS Gott koEnnten Sie mir bitte sagen ob um zirka fuEnf Uhr morgens ein Zug von MuEnchen nach Ulm geht.
568 Komme ich spaEtabends noch von Hamburg nach MuEnster.
569 Nennen sie mir bitte Abfahrtszeiten und Ankunftszeiten der ZuEge zwischen MuEnchen und WuErzburg zwischen zehn und elf Uhr vormittags
570 Wann geht Donnerstag abend der letzte Zug von MuEnchen nach Hamburg.
571 GruEsS Gott koEnnten Sie mir bitte die Abfahrtszeiten der ZuEge nennen die von Ulm nach Frankfurt fahren.
572 Welchen Zug musS ich nehmen um gegen zehn Uhr in WuErzburg zu sein.
573 Gibt es stuEndliche Verbindungen zwischen MuEnchen und Bonn.
574 Wie lange dauert bitte eine Fahrt von MuEnchen nach Hamburg.
575 Wann faEhrt Neujahr so gegen Mittag ein Zug nach Ulm.
576 Ich moEchte gern wissen wann ich hier spaEtestens losfahren musS wenn ich zirka gegen drei Uhr nachmittags in Ulm sein musS.
577 Ich brauche fuEr morgen eine Fahrkarte nach KoEln bitte suchen Sie mir eine guEnstige Verbindung.
578 KoEnnen Sie mir eine Verbindung Augsburg Kiel geben bei der ich morgens losfahren kann.
579 Bitte machen sie mir VorschlaEge wie ich am schnellsten nach Augsburg komme
580 Ich braEuchte eine Auskunft und zwar welche ZuEge fahren heute abends von Ulm nach Frankfurt.
581 FaEhrt immer noch um acht Uhr vierundzwanzig ein Intercity nach Mannheim.
582 Ich musS nach Hamburg aber ich moEchte nicht in der Nacht ankommen.
583 Welche MoEglichkeiten habe ich samstags vormittags nach Augsburg zu fahren.
584 Ich will morgen nach Hamburg fahren. Wann musS ich ungefaEhr losfahren wenn ich zwischen sieben und acht Uhr abends dort sein moEchte.
585 Nach Regensburg Dienstag morgen gegen acht Uhr wann fahren da ZuEge.
586 Ich moEchte nach Ulm und moEglichst am Wochenende fahren.

587 KoEnnen sie mir sagen wann ich hier wegfahren musS wenn ich spaEtestens morgen fruEh um halb acht in Hamburg sein will.
588 Ich benoEtige Auskunft uEber ankommende ZuEge.
589 KoEnnen sie mir auch Auskunft geben uEber ZuEge die nach Hamburg fahren.
590 Von MuEnchen nach KoEln uEber Augsburg Ankunft in KoEln gegen drei Uhr nachmittags.
591 Ich will nach MuEnchen fahren und zwar von DuEsseldorf aus morgen fruEh geben Sie mir die Verbindungen.
592 Ich moEchte nach WuErzburg und von dort aus nach Berlin.
593 Suche einen Zug von hier nach Ulm der vor siebzehn Uhr dort ankommt.
594 Welche ZuEge fahren zwischen acht und zehn Uhr uEber den Feiertagen nach Ulm.
595 Nach Dortmund bitte ich moEchte Dienstag morgen um neun Uhr da sein.
596 Ich moEchte mit dem Zug nach Hamburg und zwar mit dem naEchsten Zug.
597 Ich moEchte so nach Bremen fahren dasS ich gegen siebzehn Uhr ankomme.
598 Ich wollte fragen ob der Zug von Paris nach MuEnchen einen Schlafwagen dabei hat.
599 KoEnnten sie mir sagen wann in der Woche vormittags zwischen zehn und elf Uhr ZuEge Richtung Frankfurt fahren.
600 Verkehrt der acht Uhr dreiundzwanzig Zug nach Frankfurt auch an Heiligabend.
601 Ich musS am Rosenmontag nach KoEln.
602 Wie bitte.
699 Ich moEchte gerne nach Paris.
700 Kann man auch ohne Umsteigen fahren.
701 ich haEtte gerne eine Zugverbindung fuEr morgen von Regensburg nach Dortmund uEber KoEln.
702 morgens zwischen acht und neun also nach acht.
703 ich haEtte gern eine Zugverbindung nach Hamburg und zwar mit Intercity oder Eurocity ohne Umsteigen.
704 morgens gegen zehn jedenfalls nicht nach nicht vor neun.
705 ich brauch eine Zugverbindung Etterzhausen Hindelang am Samstag vormittag hin und am Sonntag moEglichst spaEt abends zuruEck.
706 Ankunftszeit in Hindelang oder Abfahrtszeit in Etterzhausen.
707 na das geht.
708 geht von Etterzhausen nach Hindelang uEberhaupt eine Bahnverbindung.
709 ich moEcht entweder Freitag abend oder Samstag sehr fruEh dann in Hindelang sein.
710 mir fehlt noch die Verbindung fuEr Samstag fruEh.
711 eine Zugverbindung bitte nach Oldenburg wobei der Zug spaEtestens um neun Uhr ankommen sollte.
712 geht es nicht eher.
713 aber sicher.
714 ja dann nehme ich den um zehn Uhr vierundzwanzig.
715 ich braEuchte eine Zugverbindung von Regensburg nach Hamburg bei der man moEglichst wenig umsteigen musS.
716 wieviel Stunden faEhrt man denn da.

717 ja dann so am Vormittag am spaEteren Vormittag vielleicht.
718 Ohne. Ohne Umsteigen.
719 also ich moEchte eine Zugverbindung von Regensburg nach Dortmund uEber KoEln mit mindestens zwei Stunden Aufenthalt in KoEln.
720 neun Uhr.
721 kann ich nach Dortmund uEber KoEln fahren.
722 Ja gut.
723 guten Tag ich moEchte eine Auskunft und zwar eine Verbindung von Regensburg nach Hamburg.
724 ah ja am Vormittag so um acht Uhr vielleicht.
725 man braucht dann von NuErnberg nach Hamburg nicht mehr umzusteigen nein.
726 gibt es eine Zugverbindung heute abend nach Frankfurt und wenn ja auf welchem Gleis faEhrt der Zug ab.
727 und gibt es sonst noch Verbindungen im Laufe des Abends nach Frankfurt.
728 ja ich braEuchte einen Liegewagen von Regensburg nach Hamburg.
729 moEglichst die Fahrt uEber Nacht.
730 naja okay das pasSt schon.
731 ja am Dienstag moEchte ich nach Kiel fahren und zwar soll es da preiswerte FahrtmoEglichkeiten geben.
732 ja ungefaEhr eine Woche moEcht ich an der Ostsee bleiben.
733 ja und wie sind dann die Verbindungen.
734 naja gut. Das pasSt.
735 GruEsS Gott ich braEuchte eine Fahrkarte nach Hamburg und wollte fragen also wann der Zug abgeht dann.
736 ja am besten waEre es fruEher Nachmittag.
737 am besten waErs fruEher Nachmittag.
738 und ab welchem Gleis geht dieser Zug bitte.
739 danke.
740 ich moEchte am dreiundzwanzigsten zwoElften nach Oldenburg fahren und zwar moEchte ich in Oldenburg fruEh sein wenn moEglich vor neun Uhr.
741 nein das ist mir dann zu spaEt.
742 ja das ist zu fruEh.
743 dann nehm ich lieber den spaEteren.
744 ja gut das wars dann vielen Dank.
745 ich moEchte gerne eine Zugverbindung von Regensburg nach KoEln und dann moEchte ich gerne weiter nach Dortmund fahren.
746 gegen acht Uhr wenns geht.
747 naja etwa vier Stunden.
748 und wann kann ich dann weiter nach Dortmund fahren.
749 ja.
750 ja gut prima danke.
751 koEnnten Sie mir bitte ZuEge von Regensburg nach Frankfurt heute abend sagen.
752 ja und geht da spaEter auch noch ein Zug.
753 das ist dann die einzige andere MoEglichkeit die dann noch ist.
754 und koEnnten Sie mir bitte noch sagen auf welchen Gleisen die ZuEge abfahren.
755 von beiden ZuEgen.

756 gut danke.
757 ich braEuchte eine Zugverbindung von Regensburg nach Hamburg moEglichst ohne Umsteigen.
758 ab zehn Uhr vormittags.
759 koEnnten Sie mir bitte sagen welche ZuEge heute abend von Regensburg nach Frankfurt gehen.
760 koEnnten Sie mir bitte sagen welche ZuEge heute abend von Regensburg nach Frankfurt gehen und an welchem Bahnsteig.
761 Gibt es noch andere MoEglichkeiten.
762 also ich brauche eine Fahrkarte nach Mannheim die wenn moEglich auch fuEr die ganze naEchste Woche gilt.
763 gelten diese Fahrkarten fuEr mehrere Tage.
764 fuEr heute abend brauche ich alle Abfahrtszeiten der ZuEge von Regensburg nach Frankfurt und die Gleise.
765 und spaEter faEhrt keiner mehr. ich brauche nur die Abfahrtszeit.
766 ich brauche fuEr den zweiundzwanzigsten Dezember einen Zug nach Oldenburg gegen Abend.
767 ich moEchte aber ungefaEhr um zweiundzwanzig Uhr in Oldenburg sein.
768 wann geht bitte ein Zug nach Hamburg mit moEglichst wenig ja BahnhoEfen.
769 wann geht bitte naEchste Woche ein Zug von Regensburg nach Hamburg am Vormittag.
770 auf welchem Gleis geht der Zug weg von Regensburg aus.
771 ich braEuchte eine Verbindung die von Regensburg nach NuErnberg geht und zwar vormittags.
772 und welche Zugverbindung gibt es von NuErnberg nach Regensburg und das nach achtzehn Uhr.
773 gibt es auch eine Verbindung gegen zwanzig Uhr.
774 ich brauche eine Zugverbindung Regensburg Dortmund mit einem Zwischenstop in KoEln.
775 morgens um acht.
776 gibt es einen Zug der in KoEln laEnger haElt und nach Dortmund weiterfaEhrt.
777 ungefaEhr eine Stunde.
778 wann geht bitte morgen ein Zug von Regensburg nach Oldenburg so dasS ich in Oldenburg um neun Uhr morgens ankomme.
779 ja wann geht der Zug bitte ab von Regensburg.
780 hat der Zug Schlafwagenabteile.
781 ich braEuchte eine Verbindung fuEr Dienstag von hier nach Kiel an der Ostsee.
782 ich wuErde gerne gegen zehn Uhr vormittags fahren.
783 was ist die billigste MoEglichkeit fuEr ein Bahnticket von hier nach Kiel.
784 nein.
785 hin und zuruEck.
786 das kann ich nicht genau sagen.
787 wie lange und an welchen Tagen ist dieses Ticket guEltig.
788 ich braEuchte die Abfahrtszeiten aller ZuEge heute von Regensburg nach Frankfurt.

789 ich braEuchte die Abfahrtszeiten aller ZuEge nach Frankfurt ab
achtzehn Uhr mit Angabe des Bahngleises.
790 ich benoEtige die Zugverbindung zwischen NuErnberg und Regensburg
ab achtzehn Uhr am Samstag.
791 ich brauche Zugverbindung NuErnberg Regensburg nach zwanzig Uhr
Samstag.
792 ich brauche die Abfahrtszeiten und Bahnsteige der Zugverbindungen
Regensburg nach Frankfurt am Mittwoch ab neunzehn Uhr.
793 also GruEsS Gott ich moEchte heute nach NuErnberg fahren und zwar
wollt ich dann fragen wann morgens ein Zug faEhrt.
794 ich brauche eine ZugfahrtmoEglichkeit erste Klasse hin und zuruEck
ab Regensburg nach Hamburg.
795 ich moEcht gern wissen wann morgen ein Zug von Regensburg nach
NuErnberg geht am Morgen und wann er zuruEckgeht.
796 ich moEcht gern wissen wann morgen ein Zug nach NuErnberg geht
so um siebzehn Uhr oder zwischen einundzwanzig und zweiundzwanzig Uhr.
797 wann faEhrt bitte ein Zug von Regensburg nach Dortmund am
Montag am Montag morgen.
798 wann faEhrt bitte ein Zug am Montag morgen nach KoEln von
Regensburg aus und in Frankfurt will ich zwei Stunden Aufenthalt haben.
799 wann fahren bitte ZuEge nach Frankfurt und auf welchem Gleis
fahren die ZuEge ab.
800 wann fahren bitte heute Abend ZuEge von Regensburg nach Frankfurt
zwischen sechs Uhr und neun Uhr am Abend.
801 Ich musS am Rosenmontag nach KoEln.
802 Wie bitte.

Diese Arbeit ist von mir selbst verfaßt worden. Ich habe keine anderen als die angegebenen Hilfsmittel verwendet.

Hans H. Weber