Aus: Computational Linguistics. An International Handbook on Computer Oriented Language Research and Applications. Edited by I. S. Bátori, W. Lenders, W. Putschke. Berlin: De Gruyter 1989. (Handbücher zur Sprach- und Kommunikationswissenschaft.) S. 378-431.

32. Parsing natürlicher Sprachen: Realisierungen

Der Artikel 31: Grundlagen des Parsings natürlicher Sprachen wird im folgenden vorausgesetzt. Begriffe, die im Artikel 31 näher erläutert werden, sind mit * gekennzeichnet. Verglichen mit den tatsächlichen Phänomenen natürlicher Sprachen, sind die folgenden Beispiele stark vereinfacht. Statt einfacher Kategorien müssen in Wirklichkeit durchweg grammatische Merkmalsmengen miteinander in Übereinstimmung gebracht werden. Für viele Erscheinungen gibt es überhaupt noch keine generellen Analyseverfahren.

1. Ableitungsorientierte Phrasenstrukturparser

Bei den folgenden Verfahren besteht die *Analysestrategie darin, die Ableitung der zu analysierenden Symbolkette aus einem Anfangssymbol nach den *Ersetzungsregeln einer *Konstituentenstrukturgrammatik zu rekonstruieren. Falls dies gelingt, wird die Eingabe akzeptiert. Die Ausgabe ist ein Konstituentenstrukturbaum, dessen Verzweigungen der Anwendung der Regeln entsprechen.

1.1. Top-down Parser

Die Rekonstruktion der Ableitung beginnt mit dem Anfangssymbol, das die Konstituente bezeichnet, welche von der gesamten Eingabe gebildet wird. Nach den Regeln der Grammatik werden die Symbole für Konstituenten fortgesetzt durch Symbolfolgen für deren Teile ersetzt, bis die der Eingabe zugeordnete Kette lexikalischer Kategorien abgeleitet ist. Die Regeln werden also von links nach rechts angewandt; der gesuchte Konstituentenstrukturbaum wird von oben nach unten entwickelt. (Unter lexikalischen Kategorien verstehen wir die syntaktischen Kategorien, die dem Lexikon entnommen werden, also in der Regel Wortarten und grammatische Merkmale; in der Theorie der formalen Sprachen spricht man von prä-terminalen Symbolen.)

1.1.1. Top-down Parser mit Rücksetzen

Prinzip: Beginnend mit dem Anfangssymbol der Grammatik wird durch Expansion der Kategorien, von oben nach unten, *Tiefe zuerst, eine *Linksableitung erzeugt. Immer, wenn als äußerst linke Konstituente eine lexikalische Kategorie abgeleitet worden ist, wird diese mit der lexikalischen Kategorie des nächsten Elements in der Eingabe verglichen. Stimmen beide überein, wird in der Eingabe um eins weiter gerückt und in der Ableitung zum nächsten Symbol eine Expansion gesucht. Die Eingabe wird so von links nach rechts in einem Durchgang durchlaufen. Gibt es für ein in der Ableitung anstehendes Symbol alternative Expansionen, so wird zunächst die erste Alternative soweit wie möglich weiter verfolgt. Ist zu einem anstehenden Symbol keine Expansion mehr möglich und deckt es sich auch nicht mit der Kategorie des Eingabeelements an der aktuellen Position, so wird die nächste der zuletzt aufgetretenen Alternativen ausprobiert (schematisches *Backtracking). Dabei muß auch zu der zur Zeit der Alternative aktuellen Position in der Eingabe zurückgegangen werden. (Aho/Ullman 1972, 289ff.; Mayer 1978, 141ff.; Klenk/Mau 1979; Winograd 1983, 94.)

Voraussetzungen:

(V-1) eine dem Anfangssymbol der Grammatik entsprechende *Vorsegmentierung des Eingabetextes (also z. B. Eingabe satzweise).

(V-2) ein Lexikon, mit dessen Hilfe den Elementen der Eingabe (in der Regel den Wörtern) grammatische Kategorien zugewiesen werden (in der *Lexikonphase).

(V-3) ein Eingabebereich, in dem die den Elementen zugewiesenen Kategorien in der Reihenfolge der Eingabe gespeichert werden.

(V-4) eine Variable, die die laufende Position in der Eingabe bezeichnet und zu Be-

ginn den Wert 1 hat, sowie eine Variable, die die Zahl der eingelesenen Elemente enthält.

(V-5) die Menge der Ersetzungsregeln einer kontextfreien Phrasenstrukturgrammatik ohne Linksrekursivität. (Bei dem folgenden Verfahren ist eine linksrekursive Regel pro Kategorie allerdings nicht schädlich, wenn ihr die nicht-rekursiven Regeln vorangehen).

(V-6) ein Arbeitsbereich, der (in der jeweils letzten Zeile) die noch zu verarbeitenden Symbole der Ableitung in der Ordnung von links nach rechts enthält.

GRAMMATIK G1

Regeln:

(R-1)	S	→	NP VP
(R-2)	VP	→	vi
(R-3)	VP	→	vt NP
(R-4)	VP	→	vt NP PP
(R-5)	NP	→	n
(R-6)	NP	→	det n
(R-7)	NP	→	det adj n
(R-8)	PP	→	präp NP

Lexikon:

vi	=	{rechnen, antworten}
vt	=	{verarbeiten, erzeugen}
det	=	{die, keine}
adj	=	{beliebigen}
n	=	{computer, eingaben, regeln, antworten, disketten}
präp	=	{auf, nach}

EINGABEBEREICH:

Eingabe:	computer	verarbeiten	eingaben	nach	regeln
Lexikon:	n	vt	n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

ARBEITSBEREICH:				**RÜCKSETZSPEICHER:**			
P:	**B:**	**A:**	**Erklärung:**	**B:**	**P:**	**A:**	**R:**
1	0	(1) S	Anfangszustand				
1	0	(2) NP VP	Expansion R-1				
1	1	(3) n VP	Expansion R-5	(1)	1	2	R-6
2	1	(4) VP	Erkennen n				
2	2	(5) vi	Expansion R-2	(2)	2	4	R-3
2	1	(4) VP	Rücksetzen B = 2				
2	2	(5) vt NP	Expansion R-3	(2)	2	4	R-4
3	2	(6) NP	Erkennen vt				
3	3	(7) n	Expansion R-5	(3)	3	6	R-6
3	3	(8) —	Erkennen n				
3	2	(6) NP	Rücksetzen B = 3				
3	3	(7) det n	Expansion R-6	(3)	3	6	R-7
3	2	(6) NP	Rücksetzen B = 3				
3	2	(7) det adj n	Expansion R-7				
2	1	(4) VP	Rücksetzen B = 2				
2	1	(5) vt NP PP	Expansion R-4				
3	1	(6) NP PP	Erkennen vt				
3	2	(7) n PP	Expansion R-5	(2)	3	6	R-6
4	2	(8) PP	Erkennen n				
4	2	(9) präp NP	Expansion R-8				
5	2	(10) NP	Erkennen präp				
5	3	(11) n	Expansion R-5	(3)	5	10	R-6
6	3	(12) —	Erkennen n ein Ergebnis gefunden				

Legende: P = laufende Position im Eingabebereich, A = Zeilen im Arbeitsbereich, B = oben auf dem Stapel befindlicher Eintrag im Rücksetzspeicher, R = nächste alternative Regel.

Abb. 32.1: Top-down Parser mit Rücksetzen

Das am weitesten links stehende Symbol (in dieser Zeile) ist das als nächstes zur Verarbeitung anstehende Symbol. Zu Beginn enthält dieser Bereich nur das Anfangssymbol der Grammatik (in der ersten Zeile).

(V-7) ein Rücksetzspeicher, d. i. ein *Stapelspeicher, in dem im Falle von Alternativen die laufende Position im Eingabebereich, der Zustand des Arbeitsbereichs (notiert in Form der aktuellen Zeilennummer) und die nächste der alternativen Regeln festgehalten wird.

Algorithmus:

(A-1) Expansion: Ist kein anstehendes Symbol vorhanden, so weiter (A-4). Steht das anstehende Symbol im Arbeitsbereich auf der linken Seite einer Regel, so ersetze es (in einer Kopie auf einer neuen Zeile) durch die Folge der Symbole auf der rechten Seite der Regel (dadurch wird das erste Symbol dieser Folge das neue anstehende Symbol im Arbeitsbereich). Steht das anstehende Symbol auf der linken Seite einer weiteren Regel, so mache einen neuen Eintrag im Rücksetzspeicher, bestehend aus der laufenden Position, der aktuellen Zeile im Arbeitsbereich und der Nummer dieser alternativ anwendbaren Regel. Wurde eine Regel gefunden, so weiter (A-1). Sonst weiter (A-2).

(A-2) Erkennen: Stimmt die Kategorie des anstehenden Symbols im Arbeitsbereich mit der Kategorie an der laufenden Position im Eingabebereich überein, so entferne (in einer Kopie auf einer neuen Zeile) das anstehende Symbol (dadurch wird das nächste Symbol in der Folge das anstehende), erhöhe die laufende Position im Eingabebereich um 1, und weiter (A-3). Sonst weiter (A-4).

(A-3) Erfolgsprüfung: Ist die laufende Position nicht größer als die Zahl der Elemente in der Eingabe, so weiter (A-1). Überschreitet die laufende Position die Zahl der Elemente und steht im Arbeitsbereich kein Symbol mehr an, so akzeptiere die Eingabe. Sollen alle Strukturen einer möglicherweise mehrdeutigen Eingabe ermittelt werden, so weiter (A-4), andernfalls beende die Prozedur. Stehen im Arbeitsbereich noch Symbole an, so weiter (A-4).

(A-4) Rücksetzen: Ist der Rücksetzspeicher leer, so weiter (A-5). Sonst ersetze die laufende Position, die aktuelle Zeile im Arbeitsbereich sowie die nächste zu benutzende Regel durch die entsprechenden Werte des jüngsten Eintrags im Rücksetzspeicher. Entferne diesen Eintrag vom Rücksetzspeicher. Damit ist der Zustand im Arbeitsbereich vor der Anwendung der letzten Regel wieder hergestellt. Weiter (A-1).

(A-5) Mißerfolg: Ist zu keiner Zeit die Erfolgsbedingung in (A-3) erfüllt gewesen, so weise die Eingabe zurück. Beende die Prozedur.

Anmerkung. Der Arbeitsbereich ist eine Art Protokoll der Schritte des Parsers. Beim Rücksetzen der Laufvariable A auf einen früheren Wert bleiben alle vorangehenden Zeilen erhalten, alle folgenden werden überschrieben, d. h. die entsprechenden Expansionen sind annulliert. Der Rücksetzspeicher wird selbst zurückgesetzt. Man sieht an dem Beispiel, wie kostspielig die top-down Analyse mit schematischem Backtracking ist. Die Analyse der zweiten Nominalphrase in (8) war richtig. Bevor jedoch zur alternativen Expansion der Verbalphrase nach (4) zurückgesetzt wird, werden erst alle anderen Ersetzungen von NP durchprobiert. Später muß die Nominalphrase noch einmal expandiert werden. Der Leser mag ausrechnen, wie viele Schritte der Parser im Anschluß an die oben dokumentierten Zustände noch ausführen müßte, bis der Rücksetzspeicher leer ist und damit feststeht, daß es keine weitere Lesung für den eingegebenen Satz gibt.

Kongruenz. Zur Berücksichtigung der Kongruenz in Kasus, Numerus, Person usw. müßten *komplexe Kategorien in den Regeln verwendet werden. Im Erkennungsschritt müßte die anstehende Kategorie zunächst mit der ebenfalls komplexen Kategorie des Eingabeelementes *unifiziert werden. Bevor die anstehende Kategorie vom Arbeitsspeicher entfernt wird, müßten die resultierenden Merkmalsmengen an die identischen Parameter der verbleibenden Kategorien weitergegeben werden (also z. B. der Numerus von n in (3) an die VP). Da lexikalische Kategorien häufig inkompatible Kreuzklassifikationen darstellen (z. B. ist „computer" entweder Singular und Nominativ, Dativ, Akkusativ oder Plural und Nominativ, Genitiv, Akkusativ), müßte es auch einen Rücksetzmechanismus für den Erkennungsschritt geben.

*Herstellung der *Strukturbeschreibung.* Die Folge der erfolgreich angewendeten Produktionen kann dazu benutzt werden, um einen Phrasenstrukturbaum zu erzeugen. Im obigen Beispiel werden die Regeln 1, 5, 4, 5, 8, 5 erfolgreich angewendet. Die Nummernfolge bildet ein Kennwort (ein sog. Parse). Auf Grund dessen wird ein *Baumgraph bzw. eine *Liste konstruiert, indem, mit der ersten Regel angefangen, jeweils die Symbole, die sich auf der rechten Seite der Regeln finden, dem Symbol auf der linken Seite untergeordnet werden, wobei letzteres jeweils das bis dahin am weitesten oben links befindliche nicht-lexikalische Symbol im Baum ist. Eine zweite Möglichkeit ist es, die Strukturbeschreibung schon während der Analyse zu erzeugen. Man organisiert dazu den Arbeitsbereich selbst als Folge von Bäumen bzw. Listen, wobei die Listen, bei denen die Erfolgsprüfung positiv ausging, zugleich die Ergebnisse des Parsings darstellen. (Vgl. De Roeck 1983.) Zu Anfang enthält der Arbeitsbereich eine Liste, deren einziges Element das Startsymbol der Grammatik ist. Dies ist zunächst auch der aktuelle Knoten. Zur Kategorie des aktuellen Knotens wird eine Produktionsregel gesucht. Statt nun die Kategorie im Arbeitsbereich durch die Symbole auf der rechten Seite der Regel zu ersetzen, werden letztere dem aktuellen Knoten in der Liste als Elemente untergeordnet. Anschließend wird das erste dieser Elemente zum aktuellen Knoten. Ist ein lexikalischer Knoten erreicht, wird das nächste nicht-lexikalische Element im Baum zum aktuellen Knoten. Beim Zurücksetzen wird eine frühere Listenfolge und darin ein anderer Knoten wieder aktuell. Der Vorteil dieses Vorgehens ist vor allem, daß auf diese Weise komplexe Kategorien flexibler in Übereinstimmung gebracht werden können, und so die Kongruenzverhältnisse einfacher zu handhaben sind.

Parsing als Beweisen. Der Idee des logischen Programmierens zufolge (siehe Kowalski 1979) ist ein Programm die Beschreibung eines Problembereichs in Form prädikatenlogischer Aussagen zusammen mit einem Verfahren zur automatischen Deduktion. Das maschinelle Erkennen eines Satzes kann man unter dieser Voraussetzung als Beweisprozedur auffassen. Die Regeln der Grammatik bilden eine Menge von Axiomen, aus denen als Theorem abzuleiten ist, daß die Eingabe ein Satz ist. Zur Realisierung des logischen Programmierens wurde die Sprache PROLOG entwickelt; eine Grammatik in Form eines PROLOG-Programms ist die Definite Clause Grammar (Pereira/Warren 1980, vgl. Gust 1983). Der Algorithmus, dem das PROLOG-Programm folgt, entspricht dem top-down Parsing mit schematischem Rücksetzen. Das Verfahren ist also unter Programmiergesichtspunkten nicht sehr effizient. Der Vorteil einer PROLOG-Implementierung besteht im eingebauten Mechanismus der *Unifikation beliebiger Merkmale und der großen Flexibilität, mit der während des Parsens neue Listen aufgebaut werden können.

1.1.2. Top-down Parser mit paralleler Abarbeitung

Prinzip: Alle Regeln, die auf ein an der laufenden Position anstehendes Symbol passen, werden angewendet. Für jede der entstehenden Symbolketten wird wieder eine Linksableitung versucht. Dieser Prozeß wird solange fortgesetzt, bis überall lexikalische Kategorien anstehen. Sodann werden die Ableitungen ausgewählt, deren anstehendes Symbol mit der Kategorie des Eingabeelementes in der entsprechenden Position übereinstimmt. Dieses Element ist damit erkannt. Es wird in der Eingabe um eins vorgerückt, und die restlichen Symbole jeder der parallelen Ableitungen werden wiederum expandiert. (Winograd 1983, 103; Klenk 1983.)

Voraussetzungen:

(V-1) bis (V-5) wie in 1.1.1. beim top-down Parser mit Rücksetzen. Linksrekursion ist nicht erlaubt.

(V-6) ein Arbeitsbereich, der die Menge der noch zu verarbeitenden Symbolketten aller alternativen Ableitungen enthält. Die am weitesten links stehenden Symbole sind die anstehenden. Zu Beginn enthält der Bereich nur das Anfangssymbol der Grammatik.

Algorithmus:

(A-1) Expansion: Ersetze jede Ableitung im Arbeitsbereich, in der ein nicht-lexikalisches Symbol ansteht, durch die Menge der Ableitungen, welche sich durch die Expansion dieses Symbols nach allen möglichen Regeln ergibt, und weiter (A-1). Gibt es keine nicht-lexikalischen anstehenden Symbole mehr im Arbeitsbereich, so weiter (A-2).

(A-2) Erkennen: Entferne aus der Menge der Ableitungen im Arbeitsbereich alle

diejenigen, deren anstehendes Symbol nicht mit der Kategorie an der laufenden Position im Eingabebereich übereinstimmt. Bleibt keine Ableitung übrig, so weiter (A-4). In den verbleibenden Ableitungen entferne das anstehende Symbol (dadurch wird das nächste Symbol in jeder Ableitung zum anstehenden), erhöhe die laufende Position um 1, und weiter (A-3).

(A-3) Erfolgsprüfung: Ist die laufende Position nicht größer als die Zahl der Elemente in der Eingabe, so weiter (A-1). Ist der Arbeitsbereich leer, so akzeptiere die Eingabe und beende die Prozedur. Sonst weiter (A-4).

(A-4) Mißerfolg: Weise die Eingabe zurück und beende die Prozedur.

Herstellung der Strukturbeschreibung: Zu jeder der parallelen Ableitungen und ihren Folgeableitungen ist ein Kennwort aus den Nummern der verwendeten Regeln zu bilden. Aus den Kennwörtern der erfolgreichen Ableitungen werden Phrasenstrukturbäume konstruiert.

1.1.3. Prädiktive Analyse mit Rücksetzen

Prinzip: Statt erst während des Analysevorganges durch wiederholte Expansion der anstehenden Symbole zu lexikalischen Kategorien zu gelangen, werden vorab in der Grammatik die Symbole auf den rechten Seiten der Regeln solange ersetzt, bis lauter Regeln entstanden sind, deren erste unmittelbare Konstituente eine lexikalische ist. Unter Einführung von Hilfssymbolen wird außerdem dafür gesorgt, daß alle übrigen unmittelbaren Konstituenten in den Regeln nicht-lexikalische Kategorien haben. (Dies ist die sog. *Greibach-Normalform der Grammatik.)

GRAMMATIK G1 (siehe Abb. 32.1)

EINGABE: computer verarbeiten eingaben nach regeln

P:	EINGABEBEREICH:		ARBEITSBEREICH:	Erklärung:
(1)	computer	n	S	
			NP VP	Expansion S
			n VP	Expansion NP
			det n VP	Expansion NP
			det adj n VP	Expansion NP
(2)	verarbeiten	vt	VP	Erkennen $\underline{n}$
			vi	Expansion VP
			vt NP	Expansion VP
			vt NP PP	Expansion VP
(3)	eingaben	n	NP	Erkennen $\underline{vt}$
			NP PP	Erkennen $\underline{vt}$
			n	Expansion NP
			det n	Expansion NP
			det adj n	Expansion NP
			n PP	Expansion NP
			det n PP	Expansion NP
			det adj n PP	Expansion NP
(4)	nach	präp	—	Erkennen $\underline{n}$
			PP	Erkennen $\underline{n}$
			präp NP	Expansion PP
(5)	regeln	n	NP	Erkennen präp
			n	Expansion NP
			det n	Expansion NP
			det adj n	Expansion NP
			—	Erkennen n
				Erfolg

Legende: P = laufende Position im Eingabebereich.

Abb. 32.2: Top-down Parser mit paralleler Abarbeitung

Während der Analyse kann nun die Anwendung von Regeln auf solche beschränkt werden, deren linker Aufhänger mit der Kategorie des nächsten Elements im Eingabebereich identisch ist. Dies läßt sich so interpretieren, daß auf der Grundlage einer anstehenden Kategorie und einem Element in der Eingabe eine Vorhersage (prediction) darüber gemacht werden kann, welche Konstituenten folgen werden oder zumindest folgen können. In der unten verwendeten *prädiktiven Grammatik nach Kuno 1965b wird dieser Gedanke durch eine etwas geänderte Form der Regeln noch unterstrichen. (Kuno/Oettinger 1963; Greibach 1964; Kuno 1965; Dietrich/Klein 1974, 81 ff.; Kuno [1967] 1976).

Voraussetzungen:

(V-1) bis (V-4) wie in 1.1.1. beim top-down Parser mit Rücksetzen.

(V-5) eine prädiktive Grammatik, deren Regeln die folgende Form haben:

$$(A, a) \parallel v$$
$$\text{oder } (A, a) \parallel \wedge$$

wobei A ein einzelnes nicht-lexikalisches Symbol, a ein einzelnes lexikalisches Symbol und v eine Kette nicht-lexikalischer Symbole darstellen. A entspricht der Kategorie einer Produktion in Greibach-Normalform, a entspricht dem linken Aufhänger und v dem Rest der Regel. '$\wedge$' im zweiten

GRAMMATIK G2 (Greibach-Normalform, schwach äquivalent zu G1 in Abb. 32.1):

Regeln:

(R-1)	(S, n)	‖	VP
(R-2)	(S, det)	‖	N VP
(R-3)	(S, det)	‖	AN VP
(R-4)	(VP, vi)	‖	$\wedge$
(R-5)	(VP, vt)	‖	NP
(R-6)	(VP, vt)	‖	NP PP
(R-7)	(NP, n)	‖	$\wedge$
(R-8)	(NP, det)	‖	N
(R-9)	(NP, det)	‖	AN
(R-10)	(N, n)	‖	$\wedge$
(R-11)	(PP, präp)	‖	NP
(R-12)	(AN, adj)	‖	N

Lexikon:

vi	=	{rechnen, antworten}
vt	=	{verarbeiten, erzeugen}
det	=	{die, keine}
adj	=	{beliebigen}
vi	=	{computer, eingaben, regeln, antworten, disketten}
präp	=	{auf, nach}

EINGABEBEREICH:

Eingabe:	computer	verarbeiten	eingaben	nach	regeln
Lexikon:	n	vt	n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

ARBEITSSPEICHER:	**EINGABEBEREICH:**		**RÜCKSETZSPEICHER:**			
A:	**P:**	**Erklärung:**	**B:**	**P:**	**A:**	**R:**
(1) S	(1) n	Anfangszustand				
(2) VP	(2) vt	Prädiktion R-1				
(3) NP	(3) n	Prädiktion R-5	(1)	2	2	R-6
(4) $\wedge$	(4) präp	Prädiktion R-7				
(2) VP	(2) vt	Rücksetzen B = 1				
(3) NP PP	(3) n	Prädiktion R-6				
(4) PP	(4) präp	Prädiktion R-7				
(5) NP	(5) n	Prädiktion R-11				
(6) $\wedge$	(6)	Prädiktion R-7				
		Erfolg				

Legende: A = Zeilen im Arbeitsbereich, P = laufende Position im Eingabebereich, B = oben auf dem Stapel befindlicher Eintrag im Rücksetzspeicher, R = nächste Regel.

Abb. 32.3: Prädiktive Analyse mit Rücksetzen

Regelschema bedeutet, daß kein Rest vorhanden ist. Der Ausdruck links von || heißt Argumentpaar, der Ausdruck rechts davon heißt Prädiktion.

(V-6) und (V-7) wie oben beim top-down Parser mit Rücksetzen.

Algorithmus:

(A-1) Prädiktion: Bilde ein Argumentpaar aus dem anstehenden Symbol im Arbeitsbereich und der Kategorie des Elements an der laufenden Position im Eingabebereich. Gibt es eine Regel mit diesem Argumentpaar, so ersetze (auf einer neuen Zeile) im Arbeitsbereich das anstehende Symbol durch die Prädiktion dieser Regel. Gibt es eine weitere Regel mit demselben Argumentpaar, so mache einen neuen Eintrag im Rücksetzspeicher, bestehend aus der laufenden Position, der aktuellen Zeile im Arbeitsbereich und der Nummer dieser alternativen Regel. Erhöhe die laufende Position im Eingabebereich um 1 und weiter (A-3). Gibt es keine passende Regel, aber eine alternative Kategorie für das Eingabeelement, so weiter (A-1), sonst weiter (A-4).

(A-3) Erfolgsprüfung,

(A-4) Rücksetzen, und

(A-5) Mißerfolg wie oben beim top-down Parser mit Rücksetzen. (A-2) Erkennen entfällt, da das jeweilige Eingabesymbol schon bei der Prädiktion berücksichtigt wird.

Anmerkung: Die Argumentpaare der prädiktiven Grammatik stellen eine Verbindung zwischen top-down und bottom-up Informationen dar. Die prädiktive Analyse ist also zugleich *erwartungs- und *datengesteuert. Das macht sie relativ effizient. Statt der zweiundzwanzig Symbol-Ersetzungen, die der top-down Erkenner mit Rücksetzen oben vornimmt, bevor das erste Ergebnis feststeht, sind bei der prädiktiven Analyse nur acht notwendig (und es steht fest, daß es keine weitere Analyse gibt); statt viermal braucht der Automat in unserem Beispiel nur einmal zurückzusetzen. Zwar wird eine Grammatik in Greibach-Normalform sehr umfangreich (Kuno [1967] 1976 berichtet von 3500 Regeln für das Englische), aber da ein direkter Zugriff auf die passenden Regeln möglich ist, fällt ihre Menge nicht sehr ins Gewicht.

Herstellung der Strukturbeschreibung wie bei den übrigen top-down Parsern. Die Phrasenstrukturbäume, die die prädiktive Grammatik der Eingabe zuordnet, sind allerdings völlig verschieden von den Strukturbeschreibungen, die der gleichen Eingabe von der ursprünglichen kontextfreien Grammatik zugeordnet werden. In Kuno [1967] 1976 wird ein Algorithmus erwähnt, mit dem kontextfreie Grammatiken in prädiktive Grammatiken überführt und die vom Parser erzeugten Strukturbeschreibungen anschließend wieder rücktransformiert werden können.

Abwandlung. Ein prädiktiver Parser kann auch parallel arbeiten. Dies ist der Fall in der ursprünglichen Version von Kuno/Oettinger 1963. Es werden jeweils alle Prädiktionen zum jeweiligen Argumentpaar in den Arbeitsbereich geschrieben und im weiteren Verlauf diejenigen getilgt, die sich beim Vorrücken nicht erfüllen.

1.1.4. Top-down Parser mit Vorausschautabelle

Prinzip. Die Wahl der Regel, die zur Expansion des anstehenden Symbols im Arbeitsbereich verwendet wird, wird von den nächsten k Kategorien in der Eingabe abhängig gemacht.

Dabei wird eine Matrix konsultiert, welche aus der vorliegenden Grammatik vorab konstruiert worden ist. Die Zeilen dieser Matrix sind den (lexikalischen und nicht-lexikalischen) Kategorien der Grammatik zugeordnet, die Spalten den möglichen Folgen aus k Symbolen in der Eingabe. (Normalerweise wird nur k = 1 betrachtet, d. h. die Spalten entsprechen den möglichen lexikalischen Kategorien der Eingabeelemente.) Die Felder der Matrix enthalten die Angabe, ob eine und u. U. welche Regel anzuwenden ist, um von dem der Zeile zugeordneten Symbol zu der der Spalte zugeordneten Kategorie in der Eingabe zu kommen. Gibt es für jedes Paar der Matrix entweder keine oder genau eine Regel, liegt eine sog. LL(k)-Grammatik vor, d. h. unter Lesen der Eingabe von links nach rechts kann deterministisch eine Linksableitung konstruiert werden. Die LL(k)-Eigenschaft spielt eine Rolle in der Theorie des Parsens von Programmiersprachen. (Vgl. Mayer 1978, 164ff.; Aho/Ullman 1972, 334ff.; Aho/Sethi/Ullman 1986, 186ff.).

Im folgenden wird der selektive „Top-to-Bottom"-Algorithmus (STBA) nach Kuno [1967] 1976 vorgestellt. Der Matrix ist hier nur zu entnehmen, ob es überhaupt einen Weg von Symbol K_i in der Ableitung zu einer

lexikalischen Kategorie t_j gibt, oder nicht. (Griffiths/Petrick 1965; Irons 1963; Kuno 1976, 175ff.)

Voraussetzungen:

(V-1) bis (V-7) wie in 1.1.1. beim top-down Parser mit Rücksetzen.

(V-8) eine Matrix, die für jede Kategorie K_i der Grammatik eine Zeile und für jede lexikalische Kategorie t_j eine Spalte enthält. Ein Feld $(m_{i,j})$ enthält den Wert 1, wenn aus der Kategorie K_i eine Kette generiert werden kann, deren am weitesten links stehendes Symbol das lexikalische Symbol t_j ist; andernfalls ist $(m_{i,j}) = 0$.

Algorithmus:

(A-1) bis (A-5) wie in 1.1.1. beim top-down Parser mit Rücksetzen.

GRAMMATIK G1 (siehe Abb. 32.1)

Matrix	vi	vt	det	adj	n	präp
S	0	0	1	0	1	0
VP	1	1	0	0	0	0
NP	0	0	1	0	1	0
PP	0	0	0	0	0	1
vi	1	0	0	0	0	0
vt	0	1	0	0	0	0
det	0	0	1	0	0	0
adj	0	0	0	1	0	0
n	0	0	0	0	1	0
präp	0	0	0	0	0	1

EINGABEBEREICH:

Eingabe:	computer	verarbeiten	eingaben	nach	regeln
Lexikon:	n	vt	n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

P:	**A:**	**ARBEITSBEREICH:**	**Erklärung:**
1	(1)	S	
1	(2)	NP VP	Expansion R-1, Matrix (NP, n) = 1
1	(3)	n VP	Expansion R-5, Matrix (n, n) = 1
2	(4)	VP	Erkennen n Expansion R-2, Matrix (vi, vt) = 0
2	(5)	vt NP	Expansion R-3, Matrix (vt, vt) = 1
3	(6)	NP	Erkennen vt
3	(7)	n	Expansion R-5, Matrix (n, n) = 1
4	(8)	—	Erkennen n
3	(6)	NP	Rücksetzen Expansion R-6, Matrix (n, det) = 0 Expansion R-7, Matrix (n, det) = 0
2	(4)	VP	Rücksetzen
2	(5)	vt NP PP	Expansion R-4, Matrix (vt, vt) = 1
3	(6)	NP PP	Erkennen vt
3	(7)	n PP	Expansion R-5, Matrix (n, n) = 1
4	(8)	PP	Erkennen n
4	(9)	präp NP	Expansion R-8, Matrix (präp, präp) = 1
5	(10)	NP	Erkennen präp
5	(11)	n	Expansion R-5, Matrix (n, n) = 1
6	(12)	—	Erkennen n Erfolg

Legende: P = laufende Position im Eingabebereich, A = Zeilen im Arbeitsbereich.

Abb. 32.4: Top-down Parser mit Vorausschautabelle

(A-1) ergänzt um Vorausschau: Wenn eine Regel der Form $A \rightarrow W\ v$ zur Expansion des anstehenden Symbols A im Arbeitsbereich gefunden worden ist, so prüfe anhand der Matrix, ob aus W die aktuelle lexikalische Kategorie t in der Eingabe ableitbar ist, d. h. ob $(W, t) = 1$. Wenn nicht, so suche eine andere Regel.

Anmerkung: Man vergleiche das Ablaufprotokoll mit dem des top-down Parsers mit Rücksetzen. Durch die Vorausschau werden drei Sackgassen und entsprechendes Rücksetzen vermieden. Der selektive Effekt ist bei dieser einfachen Tabelle nicht ganz so groß wie bei der prädiktiven Analyse. Jedoch sind hier keine Umformungen der Regeln und Rücktransformationen der Strukturbäume nötig.

Herstellung der Strukturbeschreibung wie bei den übrigen top-down Parsern.

1.2. Bottom-up Parser

Die Rekonstruktion der Ableitung beginnt mit dem lexikalischen Elementen. Nach den Regeln der Phrasenstrukturgrammatik werden fortgesetzt unmittelbare Konstituenten zu einer Konstituente zusammengefaßt, bis das Anfangssymbol erreicht ist. Die Regeln werden also von rechts nach links angewendet. Der gesuchte Konstituentenstrukturbaum wird von unten nach oben entwickelt.

1.2.1. Shift-reduce Parser mit Rücksetzen

Prinzip: Die Eingabe wird von links nach rechts in einem Durchgang durchlaufen. Beim Vorrücken wird jeweils das nächste Eingabeelement an das Ende des Arbeitsbereichs kopiert (shift). Ist eine Kategorie am Ende des Arbeitsbereichs die letzte auf der rechten Seite einer Regel (ein 'rechter Aufhänger') und sind die übrigen Konstituenten der Regel ebenfalls vorhanden, werden die entsprechenden Symbole im Arbeitsbereich durch das Symbol auf der linken Seite der Regel ersetzt (reduce). Für dieses Symbol wird wiederum eine Regel gesucht und so fort. Ist keine Reduktion mehr möglich, wird das nächste Element der Eingabe in den Arbeitsbereich geholt. (Aho/Ullman 1972, 303 ff.; Shieber 1983 b und 1983 c.)

Voraussetzungen:

(V-1) bis (V-3) wie in 1.1.1. beim top-down Parser mit Rücksetzen.

(V-4) eine Variable, die die laufende Position in der Eingabe bezeichnet und zu Beginn den Wert 0 hat.

(V-5) die Menge der Ersetzungsregeln einer kontextfreien Phrasenstrukturgrammatik ohne Tilgungen.

(V-6) ein Arbeitsbereich, der zu Anfang leer ist; das (in der letzten Zeile) am weitesten rechts stehende Symbol im Arbeitsbereich ist jeweils das anstehende.

(V-7) ein Rücksetzspeicher, in dem bei einem Konflikt zwischen verschiedenen möglichen Aktionen des Parsers die augenblickliche Zeile im Arbeitsbereich sowie die Art des Konflikts festgehalten wird.

Algorithmus:

(A-1) Hinzufügen (shift): Erhöhe die laufende Eingabeposition um eins. Füge die Kategorie des entsprechenden Eingabeelements dem Arbeitsbereich (auf einer neuen Zeile) hinzu. Hat das Eingabeelement noch weitere Kategorien (d. h. ist es mehrdeutig), so mache einen Eintrag im Rücksetzspeicher bestehend aus der Zeile des Arbeitsbereichs vor dem Hinzufügen und einer Spezifizierung des Konflikts als 'shift/shift' (S/S). Weiter (A-2).

(A-2) Reduktion (reduce): Suche die nächste Produktion, in der das anstehende Symbol die äußerst rechte der unmittelbaren Konstituenten bildet. Wird eine solche gefunden, prüfe ob die übrigen Konstituenten in der Regel links vom anstehenden Symbol im Arbeitsbereich eine Entsprechung haben. Wenn ja, ersetze die Symbole, die den unmittelbaren Konstituenten entsprechen, (auf einer neuen Zeile) im Arbeitsbereich durch die Kategorie der Produktion. Diese Kategorie wird dadurch zum neuen anstehenden Symbol. Da statt einer Reduktion das Hinzufügen des nächsten Eingabeelements möglich gewesen wäre, außer wenn das Ende der Eingabe erreicht ist, mache einen Eintrag im Rücksetzspeicher mit der Zeile des Arbeitsbereichs vor der Reduktion und der Spezifikation 'shift/reduce' (S/R). Gibt es eine weitere Regel, die angewendet werden könnte, so besteht außerdem ein reduce/reduce-Konflikt. Mache in diesem Fall einen weiteren Eintrag im Rücksetzspeicher mit der Spezifikation 'reduce/reduce' (R/R). War keine Re-

GRAMMATIK G1 (wie in Abb. 32.1, sortiert nach rechtem Aufhänger)

Regeln:

NP →		n	(R-5)
NP →	det	n	(R-6)
NP →	det adj	n	(R-7)
VP →	vt	NP	(R-3)
PP →	präp	NP	(R-8)
VP →	vt NP	PP	(R-4)
VP →		vi	(R-2)
S →	NP	VP	(R-1)

Lexikon:

vi	=	{rechnen, antworten}
vt	=	{verarbeiten, erzeugen}
det	=	{die, keine}
adj	=	{beliebigen}
n	=	{computer, eingaben, regeln, antworten disketten}
präp	=	{auf, nach}

EINGABEBEREICH:

Eingabe:	computer	erzeugen	antworten	nach	regeln
Lexikon:	n	vt	vi/n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

ARBEITSBEREICH:			**RÜCKSETZSPEICHER:**			
P:	**A:**		**B:**	**A:**	**K:**	**Erklärung:**
1	(1)	n				Hinzufügen n
1	(2)	NP	(1)	1	S/R	Reduktion R-5
2	(3)	NP vt				Hinzufügen vt
3	(4)	NP vt vi	(2)	3	S/S	Hinzufügen vi
3	(5)	NP vt VP	(3)	4	S/R	Reduktion R-2
4	(6)	NP vt VP präp				Hinzufügen präp
5	(7)	NP vt VP präp n				Hinzufügen n
5	(8)	NP vt VP präp NP				Reduktion R-5
5	(9)	NP vt VP PP				Reduktion R-8
3	(4)	NP vt vi				Rücksetzen B = 3
4	(5)	NP vt vi präp				Hinzufügen präp
5	(6)	NP vt vi präp n				Hinzufügen n
5	(7)	NP vt vi präp NP				Reduktion R-5
5	(8)	NP vt vi PP				Reduktion R-8
2	(3)	NP vt				Rücksetzen B = 2
3	(4)	NP vt n				Hinzufügen n
3	(5)	NP vt NP	(2)	4	S/R	Reduktion R-5
3	(6)	NP VP	(3)	5	S/R	Reduktion R-3
3	(7)	S	(4)	6	S/R	Reduktion R-1
4	(8)	S präp				Hinzufügen präp
5	(9)	S präp n				Hinzufügen n
5	(10)	S präp NP				Reduktion R-5
5	(11)	S PP				Reduktion R-8
3	(6)	NP VP				Rücksetzen B = 4
4	(7)	NP VP präp				Hinzufügen präp
5	(8)	NP VP präp n				Hinzufügen n
5	(9)	NP VP präp NP				Reduktion R-5
5	(10)	NP VP PP				Reduktion R-8
3	(5)	NP vt NP				Rücksetzen B = 3
4	(6)	NP vt NP präp				Hinzufügen präp
5	(7)	NP vt NP präp n				Hinzufügen n
5	(8)	NP vt NP präp NP				Reduktion R-5
5	(9)	NP vt NP PP				Reduktion R-8
5	(10)	NP VP				Reduktion R-4
5	(11)	S				Erfolg

Legende: P = laufende Position im Eingabebereich, A = Zeilen im Arbeitsbereich, B = Zustände des Rücksetzspeichers, K = Konflikt zwischen Aktionen des Parsers.

Abb. 32.5: Shift-reduce Parser mit Rücksetzen

duktion möglich, so weiter (A-3), sonst weiter (A-2).

(A-3) Erfolgsprüfung: Ist die laufende Position noch kleiner als die Zahl der Eingabeelemente, so weiter (A-1). Enthält der Arbeitsbereich allein das Anfangssymbol der Grammatik, so akzeptiere die Eingabe. Sollen alle Strukturen einer möglicherweise mehrdeutigen Eingabe ermittelt werden, so weiter (A-4), sonst beende die Prozedur. Enthält der Bereich nicht nur das Anfangssymbol so weiter (A-4).

(A-4) Rücksetzen: Ist der Rücksetzspeicher leer, so weiter (A-5). Mache die im letzten Eintrag des Rücksetzspeichers spezifizierte Zeile des Arbeitsbereichs zur laufenden. (Damit ist der alte Zustand des Arbeitsbereichs wieder hergestellt.) Merke den Konflikt und entferne anschließend den Eintrag vom Rücksetzspeicher. Bei S/S und S/R weiter (A-1), bei R/R weiter (A-2).

(A-5) Mißerfolg: Ist zu keiner Zeit die Erfolgsbedingung in (A-3) erfüllt gewesen, so weise die Eingabe zurück. Beende die Prozedur.

Anmerkung. Die Mehrdeutigkeit von 'antworten' führt zu einem shift/shift-Konflikt in (4), der erst beim dritten Rücksetzen richtig gelöst wird. Shift/reduce-Konflikte treten bei jeder Reduktion auf, außer bei den Reduktionen am Ende der Eingabekette (weil dort nichts mehr hinzugefügt werden kann). Ein reduce/reduce-Konflikt kommt bei der gegebenen Grammatik nicht vor. Nachteilig ist, daß immer erst alle Symbole bis zum Ende der Eingabe hinzugefügt werden müssen, bevor das Rücksetzen einsetzt.

Herstellung der Strukturbeschreibung. Die beim Erkennen erfolgreich angewendeten Regeln werden anschließend in umgekehrter Reihenfolge zur Konstruktion eines Phrasenstrukturbaumes benutzt. Eine andere Möglichkeit ist es (wie in 1.2.2. und 1.2.3.), den Arbeitsbereich selbst als Folge von Listen zu organisieren.

1.2.2. Bottom-up Parser nach dem Prinzip Breite-zuerst

Prinzip: Die Folge von Kategorien der Eingebeelemente wird nach alle möglichen Regeln reduziert (*Breite zuerst). Jede Regelanwendung führt zu einer neuen Symbolfolge. (Es werden also nicht gleichzeitig mehrere Regeln auf dieselbe Symbolfolge angewandt.) Jede entstandene Symbolfolge wird wiederum nach allen möglichen Regeln reduziert, und so fort in immer neuen Durchläufen, bis in einer Sequenz das Startsymbol der Grammatik abgeleitet worden ist oder keine Reduktion mehr möglich ist. (Dietrich/Klein 1974: 70ff.)

Voraussetzungen:

(V-1) bis (V-3) wie in 1.1.1. beim top-down Parser mit Rücksetzen.

(V-4) die Menge der Ersetzungsregeln einer kontextfreien Phrasenstrukturgrammatik ohne Tilgungen.

(V-5) ein Arbeitsbereich, der aus Folgen von *Listen besteht. Die erste Folge besteht aus Listen mit den lexikalischen Kategorien. (Bei Mehrdeutigkeit von Eingabeelementen wird eine Disjunktion der Kategorien gespeichert.)

(V-6) eine Variable, die die jeweils aktuelle Folge im Arbeitsbereich angibt; zu Beginn hat sie den Wert 1.

Algorithmus:

(A-1) Reduktion: Durchlaufe die aktuelle Folge von Listen im Arbeitsbereich. Wenn das oberste Element (der 'head') der jeweiligen Liste der rechte Aufhänger in einer Produktionsregel ist und wenn die übrigen unmittelbaren Konstituenten in der Regel sich mit den obersten Elementen der vorangehenden Listen im Arbeitsbereich decken, so erzeuge eine neue Folge von Listen im Arbeitsbereich, in der die den unmittelbaren Konstituenten entsprechenden Listen zu einer neuen Liste zusammengefaßt worden sind, deren oberstes Element die Kategorie der Produktionsregel ist. Stimmt die neue Folge einschließlich aller abhängigen Elemente in den Listen mit einer schon vorhandenen Folge überein, so übergehe das Ergebnis. Sonst speichere die neue Folge im Arbeitsbereich ab. Gibt es eine weitere Regel, in der das laufende Element rechter Aufhänger ist, so verfahre ebenso und erzeuge eine weitere neue Folge. Wenn der Durchlauf das Ende der aktuellen Folge erreicht hat, so weiter (A-2).

(A-2) Neuer Durchlauf: Ist die aktuelle Folge gleich der letzten Folge im Arbeitsbereich (d. h. sind alle Regeln auf alle Folgen angewendet worden und konnte zur letzten Folge keine neue

Folge mehr erzeugt werden), so weiter (A-3). Sonst erhöhe die Variable für die aktuelle Folge im Arbeitsbereich um eins und weiter (A-1).

(A-3) Erfolgsprüfung: Ermittle alle Fälle, in denen eine Folge von Listen zu einer einzigen Liste reduziert wurde, deren oberstes Element das Startsymbol der Grammatik ist. Gib diese Liste(n) als Ergebnis aus. Existiert keine solche Liste, so weise die Eingabe zurück. Beende die Prozedur.

Strukturbeschreibung. Gemäß der Erfolgsbedingung (A-3) ist Liste (9) das Ergebnis der Analyse.

1.2.3. Tabellengesteuerter Shift-reduce Parser

Prinzip: Das Problem bei einem shift-reduce Parser besteht darin zu entscheiden, ob ein weiteres Symbol dem Arbeitsbereich hinzuzufügen oder eine Reduktion vorzunehmen ist, sowie im letzteren Fall, welche Regel zu benutzen ist. Eine effiziente Lösung dieses

GRAMMATIK G1 (siehe Abb. 32.1)

EINGABEBEREICH:

Eingabe:	computer	verarbeiten	keine	beliebigen	eingaben
Lexikon:	n	vt	det	adj	n
Position:	(1)	(2)	(3)	(4)	(5)

A:	ARBEITSBEREICH:	Ursprung:	Regel:
(1)	n vt det adj n	—	—
(2)	NP (n) vt det adj n	1	R-5
(3)	n vt det adj NP (n)	1	R-5
(4)	n vt NP (det adj n)	1	R-7
(5)	NP (n) vt det adj NP (n)	2	R-5
(6)	NP (n) vt NP (det adj n)	2	R-7
(*)	NP (n) vt det adj NP (n)	3	R-5
(*)	NP (n) vt NP (det adj n)	4	R-5
(7)	n VP (vt NP (det adj n))	4	R-3
(8)	NP (n) VP (vt NP (det adj n))	6	R-3
(*)	NP (n) VP (vt NP (det adj n))	7	R-3
(9)	S (NP (n) VP (vt NP (det adj n)))	8	R-1

Legende: A = Folgen im Arbeitsbereich; Ursprung = die zum Zeitpunkt der Erzeugung der Zeile aktuelle Folge, auf welche die neue Folge zurückgeht; Regel = die auf die aktuelle Folge angewandte Regel.

Abb. 32.6: Bottom-up Parser Breite-zuerst

Anmerkung: Die abhängigen Elemente in den Listen (d. s. diejenigen in Klammern) werden beim Vergleich mit den Regeln überlesen, bei der Prüfung, ob das Ergebnis schon vorhanden ist, aber berücksichtigt. Die mit (*) markierten Folgen werden nicht in den Arbeitsbereich aufgenommen, da sie schon existieren. Die Breite-zuerst-Analyse ist trotzdem ungünstig, weil viel Doppelarbeit geleistet wird, je länger die Eingabe ist, und weil die Reduktionen nicht durch den linken Kontext eingeschränkt werden, wie beim shift-reduce Parsing.

Problems, hängt davon ab, daß die Grenzen der Konstituenten jeweils mit Sicherheit erkannt werden. Dies versucht man zu erreichen, indem zum einen berücksichtigt wird, welche Konstituente unter Arbeit ist und wieviel der Parser davon gesehen hat. Dies ergibt eine Menge von Zuständen des Parsers. Zum anderen wird k Symbole (in der Praxis meist k = 1) vorausgeschaut. Wird der Anfang der nächsten Konstituente entdeckt, steht fest, daß eine Grenze erreicht worden ist und eine Reduktion vorgenommen werden muß. Formal wird eine Matrix aus Zuständen und le-

xikalischen Symbolen benutzt. Die Felder der Matrix enthalten Anweisungen für den Parser, nämlich Hinzufügen des Symbols zum Arbeitsbereich, Reduktion nach einer bestimmten Regel, Akzeptieren der Eingabe, oder eine Fehleranzeige (d. i. die 'action table'). Eine weitere Matrix aus Zuständen und nicht-lexikalischen Symbolen enthält in ihren Feldern die Angabe von Zuständen, zu denen gesprungen werden soll, nachdem eine Konstituente erfolgreich geparst worden ist (die 'goto table'). Lassen sich zu einer Grammatik Matrizen mit genau einem Eintrag in jedem Feld konstruieren, so liegt eine LR(k)-Grammatik vor (von links nach rechts eine Rechtsableitung erzeugend bei k Symbolen Vorausschau), für die eine deterministische bottom-up Analyse unter linearem Zeitbedarf möglich ist. (Aho/Ullman 1972, 371ff.; Mayer 1978, 264ff.; Aho/Sethi/Ullman 1986, 187ff.)

Voraussetzungen:

(V-1) bis (V-3) wie in 1.1.1. beim top-down Parser mit Rücksetzen.

(V-4) die Menge der Ersetzungsregeln einer kontextfreien Phrasenstrukturgrammatik ohne Tilgungen.

(V-5) eine Steuertabelle, bestehend aus einer Matrix aus Zuständen und lexikalischen Symbolen (der Aktionstabelle) sowie einer Matrix aus Zuständen und nicht-lexikalischen Symbolen (der Sprungtabelle). Die Felder der Tabellen enthalten Anweisungen für den Parser. Zu den lexikalischen Symbolen kommt das Zeichen '$' als Markierung des Eingabeendes hinzu.

(V-6) ein Arbeitsbereich, der als Graph vorgestellt werden kann, dessen Knoten mit Nummern von Zuständen und dessen Kanten mit Listen etikettiert sind. Die Knoten entsprechen zugleich bestimmten Punkten in der Eingabe. Die Listen sind die vorläufigen Analyseergebnisse für die Zeichenkette zwischen zwei Punkten. Zu Beginn enthält der Arbeitsbereich nur einen Knoten für den Zustand 0 und keine Kanten.

(V-7) eine Variable für den aktuellen Zustand des Parsers, die zu Beginn den Wert 0 hat.

(V-8) eine Variable, die die aktuelle Position in der Eingabe bezeichnet und zu Beginn den Wert 0 hat.

Algorithmus:

(A-1) Konsultation der Tabelle: Es wird das Feld in der Aktionstabelle aufgesucht, dessen Zeilennummer mit dem aktuellen Zustand des Parsers übereinstimmt, und dessen Spalte mit dem gleichen Symbol etikettiert ist, das sich an der nächsten Position hinter der aktuellen Position in der Eingabe befindet. Ist die nächste Position das Ende der Eingabe, so suche die Spalte mit dem Symbol '$' auf. Weiter (A-2).

(A-2) Erfolgsprüfung: Ist das Feld leer, so weise die Eingabe zurück und beende die Prozedur. Enthält das Feld den Eintrag 'accept' (abgekürzt 'acc'), so gebe die im Arbeitsbereich befindliche Liste als Ergebnis der Analyse aus und beende die Prozedur. Sonst weiter (A-3).

(A-3) Hinzufügen: Enthält das Feld die Anweisung 'shift z' (abgekürzt 'shz'), wobei z die Nummer eines Zustands ist, so bilde eine Liste aus der Kategorie des nächsten Eingabeelements und füge diese im Arbeitsbereich als Kante an den letzten Knoten an (der den letzten aktuellen Zustand repräsentiert). Mache z zum neuen aktuellen Zustand. Füge z als neuen Knoten dem Arbeitsbereich hinzu und verbinde die neue Kante mit diesem Zustand. Erhöhe die aktuelle Position um 1. Weiter (A-1).

(A-4) Reduktion: Enthält das Feld die Anweisung 'reduce r' (abgekürzt 'rer'), wobei r die Nummer einer Regel ist, so fasse die Listen im Arbeitsbereich, deren oberste Elemente den unmittelbaren Konstituenten in der Regel r entsprechen, zu einer neuen Liste unter der Kategorie der Regel zusammen. (Dabei wird im Arbeitsbereich um soviele Knoten zurückgegangen, wie Konstituenten zusammenzufassen sind.) Mache den Knoten, der der ersten der zusammengefaßten Listen (also dem 'linken Aufhänger') vorangeht, zum vorübergehenden Zustand. Hänge die neue Liste an diesen Knoten an. Suche in der Sprungtabelle die Zeile mit dem vorübergehenden Zustand auf. Entnehme der Spalte, die das Etikett der gerade gebildeten neuen Konstituenten trägt, den Zustand z. Mache z zum neuen aktuellen Zustand. Füge z als neuen Knoten dem

GRAMMATIK G1 (Regeln, nach denen reduziert wird):

(re1)	S	→ NP VP	(re5)	NP	→ n
(re2)	VP	→ vi	(re6)	NP	→ det n
(re3)	VP	→ vt NP	(re7)	NP	→ det adj n
(re4)	VP	→ vt NP PP	(re8)	PP	→ präp NP

Lexikon: wie Abb. 32.1

	AKTIONSTABELLE:							**SPRUNGTABELLE:**			
	det	adj	n	präp	vi	vt	$	NP	PP	VP	S
0	sh9		sh8					2			1
1							acc				
2					sh4	sh5				3	
3							re1				
4							re2				
5	sh9		sh8					6			
6				sh13			re3		7		
7							re4				
8					re5	re5	re5				
9		sh11	sh10								
10					re6	re6	re6				
11			sh12								
12					re7	re7	re7				
13	sh9		sh8					14			
14							re8				

EINGABEBEREICH:

Eingabe:	computer	erzeugen	antworten	nach	regeln
Lexikon:	n	vt	vi/n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

P:	**ARBEITSBEREICH:**	**Erklärung:**
0	⓪	Anfangszustand
1	⓪ —n— ⑧	shift 8
1	⓪ —NP(n)— ②	reduce 5, goto 2
2	⓪ —NP(n)— ② —vt— ⑤	shift 5
3	⓪ —NP(n)— ② —vt— ⑤ —n— ⑧	shift 8
3	⓪ —NP(n)— ② —vt— ⑤ —NP(n)— ⑥	reduce 5, goto 6
4	⓪ —NP(n)— ② —vt— ⑤ —NP(n)— ⑥ —präp— ⑬	shift 13
5	⓪ —NP(n)— ② —vt— ⑤ —NP(n)— ⑥ —präp— ⑬ —n— ⑧	shift 8
5	⓪ —NP(n)— ② —vt— ⑤ —NP(n)— ⑥ —präp— ⑬ —NP(n)— ⑭	reduce 5, goto 14
5	⓪ —NP(n)— ② —vt— ⑤ —NP(n)— ⑥ —PP(präp NP(n))— ⑦	reduce 8, goto 7
5	⓪ —NP(n)— ② —VP(vt NP(n) PP(präp NP(n)))— ③	reduce 4, goto 3
5	⓪ —S(NP(n) VP(vt NP(n) PP(präp NP(n))))— ①	reduce 1, goto 1
		accept

Abb. 32.7: Tabellengesteuerter Shift-reduce Parser

Arbeitsbereich hinzu und verbinde die neue Kante mit diesem Knoten. Weiter (A-1).

Konstruktion der Steuertabelle. Der Grundgedanke ist, daß es bei der Abarbeitung jeder Regel der Grammatik eine Menge von Zuständen gibt, die sich darin unterscheiden, wieviel von den unmittelbaren Konstituenten bereits gesehen worden sind, und wieviel noch folgen. Markieren wir die Stelle zwischen abgearbeiteten und noch abzuarbeitenden Konstituenten durch einen Punkt, so gibt es zu einer Regel der Form X → Y Z folgende Zustände:

X → .Y Z
X → Y .Z
X → Y Z.

Die Menge aller Zustände erhält man, indem man zunächst der Grammatik eine Regel hinzufügt, in der man den Zustand (Z-0) vor Beginn des Parsens des Startsymbols der Grammatik markiert, also z. B.

(Z-0) S' → .S

Derselbe Zustand, d. h. eine Abarbeitungssituation an derselben Stelle in der Eingabekette, läßt sich in alle Regeln projizieren, die zur rekursiven Expansion des Symbols hinter dem Punkt benutzt werden können. Legen wir die Grammatik G1 zugrunde, so sind im Zustand (Z-0) auch die folgenden Zustände eingeschlossen:

S → .NP VP
NP → .n
NP → .det n
NP → .det adj n

Den nächsten Zustand ermittelt man, indem man den Punkt um eine Konstituente weiter schiebt. Aus (Z-0) ergibt sich

(Z-1) S' → S.

Eine Projektion in weitere Regeln ist hier nicht möglich, da sich rechts des Punktes keine Konstituente mehr befindet. Nun fährt man fort, die übrigen Regeln in Abarbeitungsphasen zu zerlegen. Da der Zustand vor Beginn der Anwendung einer Regel schon in einer vorangehenden Projektion enthalten ist, braucht man jetzt immer erst mit dem Zustand nach Abarbeitung der ersten Konstituente zu beginnen. Für den Rest der Grammatik G1 ergeben sich die folgenden Zustände und Projektionen. (Soweit die Zerlegungen zweier Regeln der Grammatik übereinstimmen, werden sie demselben Zustand zugeordnet; vgl. unten Zustände 5, 6 und 9.)

(Z-2) S → NP .VP
VP → .vi
VP → .vt NP
VP → .vt NP PP
(Z-3) S → NP VP.
(Z-4) VP → vi.
(Z-5) VP → vt. NP
VP → vt. NP PP
NP → .n
NP → .det n
NP → .det adj n
(Z-6) VP → vt NP.
VP → vt NP. PP
PP → .präp NP
(Z-7) VP → vt NP PP.
(Z-8) NP → n.
(Z-9) NP → det. n
NP → det. adj n
(Z-10) NP → det n.
(Z-11) NP → det adj. n
(Z-12) NP → det adj n.
(Z-13) PP → präp. NP
NP → .n
NP → .det n
NP → .det adj n
(Z-14) PP → präp NP.

Die Zustände (Z-0) bis (Z-14) bilden die Zeilen der Aktionstabelle und der Sprungtabelle. Die Spalten der Aktionstabelle sind mit den lexikalischen Kategorien und '$' (für das Eingabeende) indiziert, die Spalten der Sprungtabelle mit den nicht-lexikalischen Kategorien der Grammatik. Soweit in der obigen Kollektion von Zustandsbeschreibungen ein Punkt vor einer lexikalischen Kategorie steht, wird in der entsprechenden Spalte des betreffenden Zustands in der Aktionstabelle 'shift z' eingetragen. Dabei ist z die Nummer desjenigen Zustandes, der eine Zerlegung derselben Regel darstellt, nur daß der Punkt über die lexikalische Kategorie hinweg vorgerückt ist. Soweit in der obigen Kollektion von Zustandsbeschreibungen ein Punkt vor einer nicht-lexikalischen Kategorie steht, wird in der entsprechenden Spalte des betreffenden Zustandes in der Sprungtabelle 'goto z' eingetragen. Dabei ist z die Nummer desjenigen Zustandes, der eine Zerlegung derselben Regel darstellt, nur daß der Punkt über die nicht-lexikalische Kategorie hinweg vorgerückt ist. Soweit in der obigen Kollektion von Zustandsbeschreibungen ein Punkt am

Ende einer Produktion steht, wird in der Zeile des betreffenden Zustands in der Aktionstabelle 'reduce r' eingetragen, und zwar in all denjenigen Spalten, deren Kategorie die erste lexikalische Kategorie einer Konstituente ist, die aufgrund der Regeln auf die Kategorie X folgen kann. Dabei ist r die Nummer der Produktion in der Grammatik und X die Kategorie dieser Produktion. Kommt X auch am Ende einer mit der Grammatik erzeugbaren Ableitung vor, so wird ein gleicher Eintrag in der Spalte mit dem Etikett '$' gemacht. (Zur Ermittlung der betreffenden Spalten benutzt man zwei Funktionen, FIRST und FOLLOW. FIRST ergibt zu jeder nicht-terminalen Kategorie die Menge der ersten unmittelbaren lexikalischen Konstituenten, FOLLOW ergibt die Menge der möglichen nachfolgenden Konstituenten.) In der Spalte $ der Zustände, mit denen eine S'-Produktion zu Ende geführt ist, wird 'accept' eingetragen. Die Steuertabelle zu G1 sieht demnach wie in Abb. 32.7 aus. Für eine etwas größere Grammatik wird die Tabelle maschinell hergestellt werden müssen. (Weitere Agorithmen dazu findet man in Aho/Sethi/

GRAMMATIK G3 (mehrdeutig, schließt G1 in Abb. 32.1 ein)

Regeln:

(R-1)	S	→	NP VP	(R-6)	NP	→	det n
(R-2)	VP	→	vi	(R-7)	NP	→	det adj n
(R-3)	VP	→	vt NP	(R-8)	NP	→	NP PP
(R-4)	VP	→	VP PP	(R-9)	PP	→	präp NP
(R-5)	NP	→	n				

Lexikon: wie in Abb. 32.1

Zustände:

0	S'	→	.S	10	NP	→ det. n
1	S'	→	S.		NP	→ det. adj n
2	S	→	NP. VP	11	NP	→ det n.
3	S	→	NP VP.	12	NP	→ det adj. n
4	VP	→	vi.	13	NP	→ det adj n.
5	VP	→	vt. NP	14	NP	→ NP. PP
6	VP	→	vt NP.	15	NP	→ NP PP.
7	VP	→	VP. PP	16	PP	→ präp. NP
8	VP	→	VP PP.	17	PP	→ präp NP.
9	NP	→	n.			

	AKTIONSTABELLE:							**SPRUNGTABELLE:**			
	det	adj	n	präp	vi	vt	$	NP	PP	VP	S
0	sh10		sh9					2,14			1
1							acc				
2					sh4	sh5				3,7	
3							re1				
4				re2			re2				
5	sh10		sh9					6,14			
6				re3			re3				
7				sh16					8		
8				re4			re4				
9				re5	re5	re5	re5				
10		sh12	sh11								
11				re6	re6	re6	re6				
12			sh13								
13				re7	re7	re7	re7				
14				sh16					15		
15				re8	re8	re8	re8				
16	sh10		sh9					14,17			
17				re9	re9	re9	re9				

Abb. 32.8 a: Nicht-deterministische Grammatik und Steuertabelle

EINGABEBEREICH:

Eingabe:	computer	verarbeiten	eingaben	auf	disketten
Lexikon:	n	vt	n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

P: **ARBEITSBEREICH** (Ausschnitte):

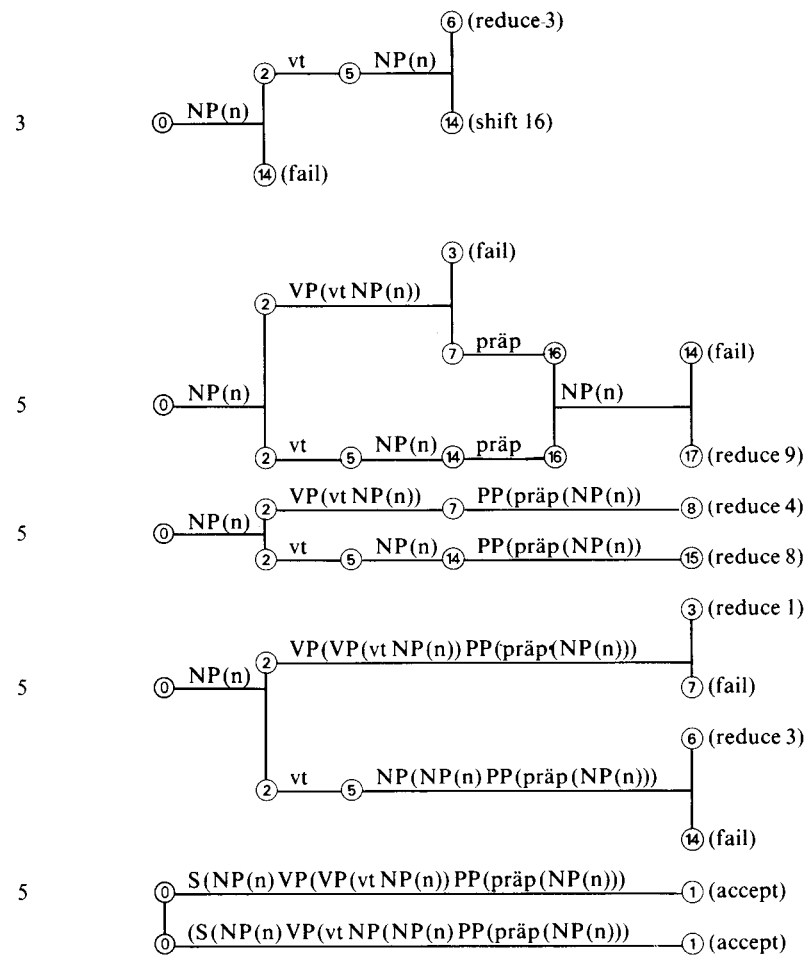


Abb. 32.8 b: Tabellengesteuerter Shift-reduce Parser für nicht-deterministische Grammatiken

Ullman 1986, 221ff.; Mayer 1978, 278ff. 353ff.)

Anmerkung. Die Effizienz des LR(k)-Parsings wird deutlich beim Vergleich mit dem shift-reduce Parser mit Rücksetzen. Es stellt sich heraus, daß der Beispielsatz mit G1 *deterministisch abgearbeitet werden kann.

Algorithmus für nicht-deterministische Grammatiken (vgl. Abb. 32.8). Eine Sprache ist nicht vom LR(k)-Typ, wenn es bei der Konstruktion der Tabellen nach dem oben geschilderten Verfahren zu Mehrfacheintragungen in ein und demselben Feld kommt. Ein shift/shift-Konflikt tritt auch auf, wenn ein Eingabeelement mehrdeutig ist und für mehr als eine Kategorie in der Tabelle Aktionen vorgesehen sind. Natürliche Sprachen zählen nicht zu den LR(k)-Sprachen. (Zu den einschlägigen Phänomenen siehe Shieber 1983.) Trotzdem braucht auf den effizienten shift-reduce Parser mit Tabelle nicht verzichtet zu werden. Die konkurrierenden Anweisungen in der Sprungtabelle von Abb. 32.8a zum Beispiel können auf einfache Weise parallel befolgt werden, indem der Graph im Arbeitsbereich an der entsprechenden Stelle zu den konkurrierenden Zuständen verzweigt. Ergeben sich für eine aktuelle Position auf verschiedenen Wegen gleiche Zustände, so kann Doppelarbeit dadurch vermieden werden, daß die Kanten zu einem Knoten zusammengeführt werden, von dem aus die Erweiterung so lange gemeinsam erfolgt, bis eine über die Zusammenführung nach links hinausgehende Reduktion den gemeinsamen Graphen wieder auflöst. Das Prinzip geht zurück auf Tomita 1987. Der Effekt ist der gleiche wie bei den unten beschriebenen Parsern mit Teilergebnistabelle.

Die Strukturbeschreibung liegt in Gestalt derjenigen Liste(n) vor, von deren Endzustand zur Anweisung 'accept' übergegangen worden ist.

2. Phrasenstruktur-Parser mit Teilergebnistabellen (chart parser)

Die Konstituentenstrukturanalyse besteht darin, die Eingabe in Segmente bestimmter Kategorie zu zerlegen, diese wiederum in Segmente, und so fort bis hin zu den kleinsten Einheiten, die durch das Lexikon vorgegeben sind. Die folgenden Verfahren sind dadurch charakterisiert, daß für eine infrage kommende Kategorie jedes Segment nur einmal ermittelt und mit seiner rechten und linken Grenze in einer Teilergebnistabelle (*chart) festgehalten wird. Bei der Analyse weiterer Segmente wird auf die bereits vorhandenen Zwischenergebnisse zurückgegriffen. Teile werden unter Berücksichtigung ihrer Grenzen auf verschiedene Weise miteinander kombiniert, ohne jedesmal wieder selbst geparst werden zu müssen.

2.1. Top-down Parser mit geteilten Produktionen

Um beim top-down Parsing Teilergebnisse wiederverwenden zu können, gilt es, für jede Stelle in der Eingabe die Konstituenten zu bestimmen, die an dieser Stelle einen bestimmten Grad der Abarbeitung erfahren haben (wobei ihr Anfang verschieden weit links liegen kann). Dies geschieht, indem für die betreffende Position eine Menge von sog. geteilten Produktionen (Mayer 1978, 111) ermittelt wird. Der eine Teil der unmittelbaren Konstituenten in einer solchen Produktion hat eine Entsprechung in den Segmenten der Eingabe gefunden. Für den Rest (remainder) steht eine Abbildung auf die Eingabe noch aus. Die Grenze zwischen dem abgearbeiteten Teil der Produktion und dem Rest wird üblicherweise durch einen Punkt markiert. (Vgl. die Produktionen bei der Konstruktion der Steuertabelle in 1.2.3.) Steht der Punkt vor allen unmittelbaren Konstituenten der Produktion, so beginnt der Vergleich erst. In diesem Fall ist die linke und die rechte Grenze des zugeordneten Segments gleich der aktuellen Position, d. h. das Segment hat noch die Länge 0. Steht der Punkt in der Mitte der Produktion, so wurde (wenn von links nach rechts vorgerückt wird) der links davon stehende Teil der unmittelbaren Konstituenten gefunden, der rechts davon stehende Teil wird noch gesucht. Die linke Grenze des zugeordneten Segmentes ist gleich der linken Grenze, die bei Beginn der Anwendung der Produktion aktuell war; die rechte Grenze ist gleich der aktuellen Position. Steht der Punkt hinter allen unmittelbaren Konstituenten in der Produktion, so ist der Vergleich komplett. Die Kategorie der Produktion kann dem entsprechenden Segment zugeordnet werden. Das Ziel des Erkenners ist es, eine komplette Produktion mit dem Startsymbol der Grammatik zu erzeugen, deren linke Grenze mit dem Anfang und deren rechte mit dem Ende der Eingabe zusammenfällt.

2.1.1. Algorithmus nach Earley

Prinzip. Die rechte Grenze der zu bildenden Segmente wird von 0 ab, d. h. vom Zustand, in dem noch nichts von der Eingabe gesehen worden ist, in Schritten von eins nach rechts verschoben, bis sie das Ende der Eingabe erreicht. Begonnen wird mit einer Menge von geteilten Produktionen für das Startsymbol der Grammatik, wobei der Punkt vor allen unmittelbaren Konstituenten steht. Nacheinander werden die geteilten Produktionen abgearbeitet, bis alle komplett sind, die sich komplettieren lassen. Typisch für den Algorithmus von Earley sind dabei drei Operationen. Der 'predictor' tritt in Aktion, wenn in einer Produktion der Punkt vor einer nichtlexikalischen Kategorie steht. Der Teilergebnistabelle werden alle Produktionen hinzugefügt, die eine Expansion der nicht-lexikalischen Kategorie darstellen. In jeder dieser Produktionen kommt der Punkt vor die erste der unmittelbaren Konstituenten. Der 'scanner' wird aufgerufen, wenn der Punkt in einer aktuellen Produktion vor einer lexikalischen Kategorie steht. Er prüft, ob sich eine analoge Kategorie an nächster Stelle in der Eingabe befindet. Wenn ja, fügt er der Teilergebnistabelle eine neue Produktion hinzu, in der der Punkt über die Kategorie nach rechts hinweg verschoben ist. Der 'completer' schließlich versucht ein komplettes Zwischenergebnis, d. i. eine Produktion mit dem Punkt am Ende, in alle anderen Produktionen einzupassen, in denen die Kategorie gerade gebraucht wird. Der Early-Algorithmus vereinigt im Rahmen einer Ableitung von links nach rechts, Tiefe zuerst, das top-down Prinzip im predictor-Schritt mit dem bottom-up Prinzip im completer-Schritt. Verglichen mit reinen top-down und reinen bottom-up Algorithmen ist er daher relativ effizient. (Earley 1970: Aho/Ullman 1972, 320ff.; Mayer 1978, 111ff.; Graham/Harrison/Ruzzo 1980.)

Voraussetzungen:

(V-1) bis (V-3) wie beim top-down Parser mit Rücksetzen (siehe 1.1.1.).

(V-4) die Menge der Regeln einer kontextfreien Phrasenstrukturgrammatik (ohne Einschränkungen).

(V-5) ein Arbeitsbereich (d. i. die Teilergebnistabelle), der eine Spalte für geteilte Produktionen, eine Spalte für die linken Grenzen und eine Spalte für die rechten Grenzen der zugeordneten Segmente umfaßt. Alle Zeilen mit derselben rechten Grenze r bilden den Teilbereich i, wobei r = i. Der Arbeitsbereich enthält in der ersten Zeile die geteilte Produktion '$\Phi \rightarrow$ .S', wobei 'S' die Startkategorie der Grammatik und 'Φ' ein zusätzliches nicht-lexikalisches Symbol ist. Die linke und die rechte Grenze für diese Produktion sind gleich 0. (Die Konstituente hinter dem Punkt in einer geteilten Produktion heiße die anstehende Konstituente.)

(V-6) zwei Laufvariablen ALT und NEU über dem Arbeitsbereich. Zu Beginn haben ALT und NEU beide den Wert 1.

Algorithmus:

(A-1) Ersetzen (predictor): Steht eine nichtlexikalische Konstituente in der Produktion der Zeile ALT an, so tue das folgende, sonst weiter (A-2). Speichere alle Produktionen der Grammatik zur Kategorie der anstehenden Konstituente in neuen Zeilen des Arbeitsbereiches (erhöhe dabei NEU). Setze den Punkt jeweils vor die erste unmittelbare Konstituente in der Produktion. Gebe der linken und der rechten Grenze in der Zeile NEU jeweils den Wert der rechten Grenze von ALT.
Prüfen auf Vorhandensein: Falls im Arbeitsbereich bereits eine identische Produktion mit denselben Werten für die linke und rechte Grenze vorhanden ist wie die aktuelle, mache den Schritt rückgängig. Weiter (A-4).

(A-2) Erkennen (scanner): Steht eine lexikalische Konstituente in der Produktion der Zeile ALT an, so tue das folgende, sonst weiter (A-3). Die aktuelle Position im Eingabebereich ist um 1 höher als die rechte Grenze von ALT. Befindet sich in dieser Position der Eingabe ein Element mit der anstehenden lexikalischen Kategorie, so kopiere die Produktion der Zeile ALT in eine neue Zeile (erhöhe dabei NEU), bewege den Punkt über die erkannte Konstituente in der Produktion hinweg, übernehme die linke Grenze in der Zeile NEU aus der Zeile ALT, setze die rechte Grenze von NEU auf den um 1 erhöhten Wert in der Zeile ALT. Weiter (A-4).

(A-3) Verwenden (completer): Ist eine Produktion in der Zeile ALT komplett, so tue das folgende. Prüfe alle bisherigen Produktionen im Arbeitsbereich, deren rechte Grenze gleich der linken Grenze

GRAMMATIK G3 (mehrdeutig, siehe Abb. 32.8)

EINGABEBEREICH:

Eingabe:	computer	verarbeiten	antworten	auf	disketten
Lexikon:	n	vt	vi/n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

ARBEITSBEREICH:

	A:	**Geteilte Produktionen:**	**LG:**	**RG:**	**Erklärung:**
Teilbereich 0:	(1)	Φ → .S	0	0	Anfangszustand
	(2)	S → .NP VP	0	0	Ersetzen (1) nach R-1
	(3)	NP → .n	0	0	Ersetzen (2) nach R-5
	(4)	NP → .det n	0	0	Ersetzen (2) nach R-6
	(5)	NP → .det adj n	0	0	Ersetzen (2) nach R-7
	(6)	NP → .NP PP	0	0	Ersetzen (2) nach R-8
	(*)	NP → .n	0	0	Ersetzen (6) nach R-5
	(*)	NP → .det n	0	0	Ersetzen (6) nach R-6
	(*)	NP → .det adj n	0	0	Ersetzen (6) nach R-7
	(*)	NP → .NP PP	0	0	Ersetzen (6) nach R-8
Teilbereich 1:	(7)	**NP → n.**	0	1	Erkennen (3) computer
	(8)	S → NP. VP	0	1	Verwenden (7) in (2)
	(9)	NP → NP. PP	0	1	Verwenden (7) in (6)
	(10)	VP → .vi	1	1	Ersetzen (8) nach R-2
	(11)	VP → .vt NP	1	1	Ersetzen (8) nach R-3
	(12)	VP → .VP PP	1	1	Ersetzen (8) nach R-4
	(13)	PP → .präp NP	1	1	Ersetzen (9) nach R-9
	(*)	VP → .vi	1	1	Ersetzen (12) nach R-2
	(*)	VP → .vt NP	1	1	Ersetzen (12) nach R-3
	(*)	VP → .VP PP	1	1	Ersetzen (12) nach R-4
Teilbereich 2:	(14)	VP → vt. NP	1	2	Erkennen (11) verarbeiten
	(15)	NP → .n	2	2	Ersetzen (14) nach R-5
	(16)	NP → .det n	2	2	Ersetzen (14) nach R-6
	(17)	NP → .det adj n	2	2	Ersetzen (14) nach R-7
	(18)	NP → .NP PP	2	2	Ersetzen (14) nach R-8
	(*)	NP → .n	2	2	Ersetzen (18) nach R-5
	(*)	NP → .det n	2	2	Ersetzen (18) nach R-6
	(*)	NP → .det adj n	2	2	Ersetzen (18) nach R-7
	(*)	NP → .NP PP	2	2	Ersetzen (18) nach R-8
Teilbereich 3:	(19)	**NP → n.**	2	3	Erkennen (15) eingaben
	(20)	**VP → vt NP.**	1	3	Verwenden (19) in (14)
	(21)	NP → NP. PP	2	3	Verwenden (19) in (18)
	(22)	**S → NP VP.**	0	3	Verwenden (20) in (8)
	(23)	VP → VP. PP	1	3	Verwenden (20) in (12)
	(24)	PP → .präp NP	3	3	Ersetzen (21) nach R-9
	(25)	**Φ → S.**	0	3	Verwenden (22) in (1)
Teilbereich 4:	(26)	PP → präp. NP	3	4	Erkennen (24) auf
	(27)	NP → .n	4	4	Ersetzen (26) nach R-5
	(28)	NP → .det n	4	4	Ersetzen (26) nach R-6
	(29)	NP → .det adj n	4	4	Ersetzen (26) nach R-7
	(30)	NP → .NP PP	4	4	Ersetzen (26) nach R-8
	(*)	NP → .n	4	4	Ersetzen (30) nach R-5
	(*)	NP → .det n	4	4	Ersetzen (30) nach R-6
	(*)	NP → .det adj n	4	4	Ersetzen (30) nach R-7
	(*)	NP → .NP PP	4	4	Ersetzen (30) nach R-8
Teilbereich 5:	(31)	**NP → n.**	4	5	Erkennen (27) disketten
	(32)	**PP → präp NP.**	3	5	Verwenden (31) in (26)
	(33)	NP → NP. PP	4	5	Verwenden (31) in (30)
	(34)	**NP → NP PP.**	2	5	Verwenden (32) in (21)
	(35)	**VP → VP PP.**	1	5	Verwenden (32) in (23)
	(36)	PP → .präp NP	5	5	Ersetzen (33) nach R-9
	(37)	**VP → vt NP.**	1	5	Verwenden (34) in (14)
	(38)	NP → NP. PP	2	5	Verwenden (34) in (18)
	(39)	**S → NP VP.**	0	5	Verwenden (35) in (8)
	(40)	**S → NP VP.**	0	5	Verwenden (37) in (8)
	(*)	PP → .präp NP	5	5	Ersetzen (38) nach R-9
	(41)	**Φ → S.**	0	5	Verwenden (39) in (1)
	(42)	**Φ → S.**	0	5	Verwenden (40) in (1)

Legende: A = Numerierung der Teilergebnisse im Arbeitsbereich, über die die Variablen ALT und NEU laufen, LG = linke Grenze, RG = rechte Grenze; die fettgedruckten Produktionen sind komplett.

Abb. 32.9: Parsing mit dem Algorithmus von Earley

der kompletten Produktion ist, ob darin die Kategorie der kompletten Produktion als nächste Konstituente ansteht. Kopiere jede Produktion, die der Bedingung genügt, in eine neue Zeile des Arbeitsbereichs (erhöhe dabei NEU), bewege den Punkt über die gefundene Konstituente hinweg, übernehme als linke Grenze in der Zeile NEU die linke Grenze der auf diese Weise weitergeführten Produktion, setze die rechte Grenze in der Zeile NEU gleich der rechten Grenze der verwendeten Produktion in der Zeile ALT.

(A-4) Abbruchbedingung: Ist ALT gleich NEU, so ist der Arbeitsbereich erschöpft. Dann weiter (A-5). Sonst erhöhe ALT um 1 und weiter (A-1).

(A-5) Erfolgsprüfung: Befinden sich im Arbeitsbereich eine oder mehrere Produktionen '$\Phi \rightarrow$ S.' mit der linken Grenze gleich 0 und der rechten Grenze gleich der Zahl der Eingabeelemente, so akzeptiere die Eingabe und beende die Prozedur. Sonst weise die Eingabe zurück und beende die Prozedur.

Anmerkung. Die in der ersten Spalte mit (*) markierten Einträge werden rückgängig gemacht, da das entsprechende Teilergebnis schon vorhanden ist. Aufgrund dieser Festlegung ist Linksrekursivität trotz des top-down Prinzips nicht ausgeschlossen. Die Teilbereiche (einer zu jeder rechten Grenze) entsprechen den Zustandsmengen in der Originalimplementierung von Earley.

Herstellung der Strukturbeschreibung. Einen Algorithmus zum nachträglichen Aufbau eines Strukturbaumes findet man in Aho/Ullman 1972, 327ff. Günstiger ist es, die Strukturbeschreibung während des Erkennens zu erzeugen (so auch Earley 1970, 101). Insbesondere lassen sich so *Unifikationsgrammatiken mit dem Earley-Algorithmus parsen. Man betrachte dazu jede Produktion in der Grammatik sowie jede geteilte Produktion im Arbeitsbereich als Baum bzw. Liste, mit der Kategorie der Produktion als dominierendem Knoten, dem die Knoten für die unmittelbaren Konstituenten untergeordnet sind. Der Arbeitsbereich besteht demnach aus einer geordneten Menge von Listen, wobei jeder Liste neben der linken und rechten Grenze des Segments ein Zeiger auf den aktuellen Knoten beigegeben wird. Bei nichtkompletten Listen ist dies dasjenige Element, vor dem in der entsprechenden Produktion der Punkt steht. Bei kompletten Listen ist der Zeiger auf den dominierenden Knoten gerichtet. Die Menge der Listen wird unter Verwendung der Variablen ALT und NEU durchlaufen. Im Predictor-Schritt wird der aktuelle Knoten der Liste ALT mit dem dominierenden Knoten der einschlägigen Produktionen in der Grammatik unifiziert. Der Scanner unifiziert den aktuellen Knoten der Liste ALT mit der lexikalischen Kategorie des nächsten Eingabeelementes. Im Completer-Schritt werden die aktuellen Knoten der Listen, die angrenzenden Segmenten zugeordnet sind, miteinander unifiziert, wobei der Zeiger in der kompletten Liste ALT auf den dominierenden Knoten und der Zeiger in den Vorgängerlisten (deren rechte Grenze gleich der linken Grenze von ALT ist) auf das jeweils anstehende Element gerichtet sind). Infolge von Variablen innerhalb der *komplexen Kategorien in den Listen, werden bei allen drei Operationen u. U. Merkmale durch die gesamten beteiligten Bäume transferiert. Auf diese Weise wird u. a. den *Kongruenzphänomenen Rechnung getragen.

Im folgenden Beispiel sind die aktuellen Knoten der zu unifizierenden Listen unterstrichen. Die Numerierung bezieht sich auf die analogen Einträge im obigen Arbeitsbereich. Merkmale ohne Wert sind Variable, die gleich instantiiert werden müssen.

Predictor:	(2)	(S (NP Person Kasus [Nominativ]) (VP Person))
Regel (R-5)		(NP Person Kasus (n Person Kasus)
Ergebnis:	(3)	(NP Person Kasus [Nominativ] (n Person Kasus [Nominativ]))
Scanner:	(3)	(NP Person Kasus [Nominativ] (n Person Kasus [Nominativ]))
Eingabe:		(n Person [3] Kasus [Nominativ, Genetiv, Akkusativ])
Ergebnis:	(7)	(NP Person [3] Kasus [Nominativ] (n Person [3] Kasus [Nominativ]))
Completer:	(2)	(S (NP Person Kasus [Nominativ]) (VP Person))
Verwenden:	(7)	(NP Person [3] Kasus [Nominativ] (n Person [3] Kasus [Nominativ]))
Ergebnis:	(8)	(S (NP Person [3] Kasus [Nominativ] (n Person [3] Kasus [Nominativ])) (VP Person [3])

Active chart parser. Earleys Algorithmus stimmt mit dem 'active chart parser' in Winograd 1983, 116 überein. Nur die Terminologie ist eine andere. Geteilte Produktionen werden dort vorgestellt als Kanten (edges) in einem Graph (*chart), dessen Knoten (vertices) Punkte in der Eingabe (und somit linke und rechte Grenzen für die Segmente) repräsentieren. Es werden aktive und komplette Kanten unterschieden entsprechend den Produktionen mit oder ohne Rest. Aktive Kanten beginnen an einem Punkt, suchen aber noch nach der zweiten Verankerung. Komplette Kanten verbinden (als Bögen) zwei Punkte. (Vgl. auch Thompson/Ritchie 1984.)

Effizienzverbesserung. Die Ersetzungen im ersten Teilbereich hängen nur von der Grammatik, nicht vom eingegebenen Satz ab. Sie lassen sich daher vorab berechnen. Die Zahl der erzeugten Produktionen läßt sich mit Hilfe einer Vorausschautabelle oder einer Grammatik in Greibach-Normalform einschränken. Es werden nur solche Produktionen zugelassen, deren erste unmittelbare Konstituente mit der lexikalischen Kategorie des nächsten Eingabeelements übereinstimmt. Dadurch könnten 16 von den 35 Ersetzungen im obigen Beispiel vermieden werden.

Erweiterungen. Die Grammatik kann mehrere Startkategorien enthalten, die vorab in die Tabelle eingetragen werden, z. B.

(1) $\Phi \rightarrow$ S
(2) $\Phi \rightarrow$ NP

Auf diese Weise lassen sich Texte verarbeiten, die neben Sätzen selbständige Nominalphrasen, z. B. als Überschriften, enthalten.

Der Parser kann on-line betrieben werden, d. h. die Analyse kann parallel zu Eingabe erfolgen, die kontinuierlich fortschreitet. Der Scanner schließt dann das Einlesen des nächsten Eingabeelements und die Zuordnung der lexikalischen Kategorie(n) ein. Jedesmal, wenn eine Produktion mit der Kategorie Φ komplett ist, erfolgt eine Ausgabe. Der Benutzer kann die Eingabe (vom Ende her) revidieren, ohne daß die Analyse neue begonnen werden muß. Es brauchen nur die der gelöschten Eingabe entsprechenden Teilbereiche im Arbeitsbereich eliminiert zu werden.

Der Earley-Algorithmus erlaubt Tilgungsregeln. Eine Produktion 'A → Λ' (wobei 'Λ' für die leere Kette steht) angewandt auf '.A' führt sofort zu 'A.'

Ebenso kann der Algorithmus Grammatiken mit *fakultativen Konstituenten handhaben. Eine Produktion 'A → (u) v' (wobei das eingeklammerte Element fakultativ ist) angewandt auf '.A' führt zu den geteilten Produktionen 'A → .u v' und 'A → .v'. Die Regeln 5 bis 7 in G3 in Abb. 32.8 a können z. B. zusammengefaßt werden zu 'NP → (det (adj)) N'. Angewandt auf .NP ergibt dies nur zwei Einträge im Arbeitsbereich:

NP → .det (adj) N
NP → .N

Erst wenn 'det' erkannt worden ist und die Situation 'NP → det. (adj) N' eingetreten ist, werden erzeugt:

NP → det. adj N
NP → det. N

Auch Grammatiken mit Kleene Stern Notation sind möglich. Es führt jeweils

A → .[u]* v
zu
A → [.u]* v
A → .v
und A → [u.]* v
zu
A → u [.u]* v
A → u. v

*Diskonstinuierliche Konstituenten werden in manchen Formalismen mittels sog. slash-Notation beschrieben, z. B. in der Generalized Phrase Structure Grammar (vgl. Gazdar/Klein/Pullum et al. 1985, 145). Auch für diese ist der Earley-Algorithmus geeignet. In der Produktion 'A → u v w/u' bezeichnet z. B. 'u' eine Konstituente, die zur Konstituente w gehört aber nach links herausgestellt worden ist. Eine solche Produktion wird normal verarbeitet, bis die Situation 'x → y .a/a' eingetreten ist (wobei x aus w abgeleitet worden ist). In dieser Situation steht die herausgestellte Konstituente an ihrer Ursprungstelle als nächste an. Nun wird, ohne den Scanner zu bemühen, einfach zu 'x → y a/a.' übergegangen.

Abwandlung für variable Wortstellung. Shieber 1984 beschreibt eine Abwandlung des Earley-Parsers für die Behandlung variabler Wortstellung (in der GPSG). Die Produktionen der Grammatik beschreiben nur den Aufbau der Konstituenten, nicht jedoch die Abfolge der unmittelbaren Konstituenten. (Eine Produktion 'A → u v' paßt also auch zur Kette 'vu'.) Die Reihenfolge wird durch gesonderte Regeln festgelegt. Für die geteilten Produktionen gilt, daß die Symbole vor dem Punkt (also die bereits abgearbeiteten) eine Zeichenkette darstellen, die Symbole

nach dem Punkt jedoch eine Zeichenmenge. Jedes Symbol dieser Menge kann, unter Einschränkung durch die Abfolgeregeln, im Predictor-Schritt expandiert werden. Der Completer kann, wiederum unter Wahrung der Stellungsregeln, jedes Symbol dieser Menge erfüllen, wobei dieses dann in der neuen Produktion auf die Seite der Zeichenkette, also links vom Punkt, hinüberwechselt. Der Scanner schließlich prüft alle Symbole in der Menge, ob eines mit der nächsten lexikalischen Kategorie in der Eingabe übereinstimmt.

Bottom-up Initiative. Es ist vorgeschlagen worden, die Initiative für neue geteilte Produktionen dem Scanner zu übertragen (siehe Kilbury 1984). Dieser holt die lexikalische Kategorie des nächsten Elements der Eingabe in den Arbeitsbereich. Zum Beispiel

(1) n. 0 1

Der Predictor fügt nun diejenigen Produktionen hinzu, die mit einer zuvor gefundenen kompletten Konstituente beginnen, hier also

(2) NP → n. 0 1
(3) S → NP. VP 0 1
(4) NP → NP. PP 0 1

Der Completer funktioniert, wie üblich: komplette Produktionen führen offene fort. Im Prinzip ist dies eine bottom-up Analyse mit linkem Aufhänger. (Siehe 'over-the-top parsing' Kimball 1975, 'left-corner parser', Aho/Ullman 1972, 310f.) Verglichen mit dem Standardalgorithmus nach Earley fehlt der linke Anschluß zum Satzanfang hin. Damit geht aber gerade ein wesentliches Moment der *Erwartungssteuerung verloren. In jeder Position werden neue Prädiktionen angestoßen, die zum Teil unbrauchbar sind. Gibt man die Berücksichtigung des linken Kontextes auf, dann hat die Anwendungen der Regeln auch Zeit, bis der Parser einen rechten Aufhänger erkannt hat, wie in den unten beschriebenen bottom-up Parsern.

2.2. Bottom-up Parser mit Teilergebnistabelle

Die ersten Teilergebnisse für den bottom-up Parser liegen in Gestalt der Eingabeelemente vor. Aus den kleineren Segmenten werden größere Segmente gebildet, indem nach den Regeln der Grammatik unmittelbare Konstituenten zu einer Konstituente zusammengefaßt werden. Es liegt nahe, die Ergebnisse der Zusammenfassungen unabhängig voneinander zu speichern. Unter Berücksichtigung der linken und rechten Grenzen kann dann potentiell jedes Zwischenergebnis mit jedem anderen zu einer weiteren Konstituente kombiniert werden. Dadurch wird jene Doppelarbeit vermieden, die für den in 1.2.2. beschriebenen, mit der gesamten Ableitung arbeitenden Parsern charakteristisch ist.

2.2.1. Algorithmus nach Cocke

Prinzip. Die Grammatik muß in *Chomsky-Normalform vorliegen, d. h. sie kennt nur binäre Zerlegungen von Konstituenten. Beim klassischen Algorithmus nach Cocke, Kasami und Younger wird das Prinzip Breite — zuerst befolgt. Angefangen mit den lexikalischen Konstituenten werden in wiederholten Durchläufen alle möglichen Paare von Konstituenten gebildet und nach den Regeln ggf. zu einer größeren Konstituente zusammengefaßt. So entstehen Segmente von immer größerer Länge. Beim unten beschriebenen Algorithmus wird, ähnlich wie beim Earley-Algorithmus, die Position zur primären Laufvariablen gemacht. Der Parser rückt in einem einzigen Durchlauf vom Anfang bis zum Ende der Eingabe vor und ermittelt für jede Stelle alle Konstituenten (verschiedener Tiefe), die dort enden. (Hays 1967, 124ff.; Brockhaus 1971, 33ff.; Aho/Ullman 1972, 314ff.; Dietrich/Klein 1974, 75ff.; Mayer 1978, 111ff.)

Voraussetzungen:

(V-1) bis (V-3) wie beim top-down Parser mit Rücksetzen (siehe 1.1.1.).
(V-4) eine Variable, die die laufende Position in der Eingabe bezeichnet und zu Beginn den Wert 1 hat, sowie eine Variable, die die Zahl der eingelesenen Eingabeelemente enthält.
(V-5) die Menge der Regeln einer kontextfreien Phrasenstrukturgrammatik in Chomsky-Normalform (sämtliche Produktionen enthalten genau zwei unmittelbare Konstituenten).
(V-6) ein Arbeitsbereich mit einer Spalte für die Kategorie einer Konstituente, einer Spalte für die linke und einer Spalte für die rechte Grenze des zugehörigen Segments in der Eingabe, einem Zeiger auf die linke und einem Zeiger auf die rechte unmittelbare Konstituente, aus denen die Konstituente gebildet worden ist, sowie zwei Laufvariablen LFD und NEU über diesem Bereich. LFD

hat zu Beginn den Wert 1. Die Kategorie in der Zeile LFD heiße das anstehende Symbol. NEU hat den Anfangswert 0. Außerdem wird noch eine Variable ALT benötigt.

Algorithmus:

(A-1) Hinzufügen: Füge die Kategorie(n) des Elements an der laufenden Eingabeposition dem Arbeitsbereich hinzu (erhöhe dabei NEU). Setze in der Zeile

GRAMMATIK G4 (schwach äquivalent zu G1 in Abb. 32.8)

Regeln:

(R-1)	S	→	$N_{u,d}$ **V_i**
(R-2)	Vi	→	**V_t** $N_{u,d}$
(R-3)	Vi	→	**V_i** PP
(R-4)	N_d	→	det **$N_{u,a}$**
(R-5)	N_a	→	adj **N_u**
(R-6)	N_u	→	**N_u** PP
(R-7)	PP	→	**präp** **$N_{u,d}$**

Lexikon:

V_i	=	{rechnen, antworten}
V_t	=	{verarbeiten, erzeugen}
det	=	{die, keine}
adj	=	{beliebigen}
N_u	=	{computer, eingaben, regeln, antworten, disketten}
präp	=	{auf, nach}

(Anmerkung: Die Indizes sind als — teilweise alternative — grammatische Merkmale zu lesen: i = Verbalphrase ohne bzw. mit erkanntem Objekt, t = Verbalphrase mit Objekt, u = Nominalphrase ohne Artikel, d = Nominalphase mit Artikel, a = Nominalphrase mit Adjektiv.)

EINGABEBEREICH

Eingabe:	computer	verarbeiten	antworten	auf	disketten
Lexikon:	Nu	Vt	Vi/Nu	präp	Nu
Position:	(1)	(2)	(3)	(4)	(5)

ARBEITSBEREICH:

A:	**Kategorie:**	**LG:**	**RG:**	**LK:**	**RK:**	**Erklärung:**
Teilbereich 1:						
(1)	N_u	0	1	—	—	Hinzufügen computer
Teilbereich 2:						
(2)	V_t	1	2	—	—	Hinzufügen verarbeiten
Teilbereich 3:						
(3)	V_i	2	3	—	—	Hinzufügen antworten
(4)	N_u	2	3	—	—	Hinzufügen antworten
(5)	V_i	1	3	**2**	4	Reduktion nach R-2
(6)	S	0	3	1	**5**	Reduktion nach R-1
Teilbereich 4:						
(7)	präp	3	4	—	—	Hinzufügen auf
Teilbereich 5:						
(8)	N_u	4	5	—	—	Hinzufügen disketten
(9)	PP	3	5	**7**	8	Reduktion nach R-7
(10)	V_i	2	5	**3**	9	Reduktion nach R-3
(11)	N_u	2	5	**4**	9	Reduktion nach R-6
(12)	V_i	1	5	**5**	9	Reduktion nach R-3
(13)	V_i	1	5	**2**	11	Reduktion nach R-2
(14)	S	0	5	1	**12**	Reduktion nach R-1
(15)	S	0	5	1	**13**	Reduktion nach R-1

Legende: A = Nummer der Teilergebnisse, LG = linke Grenze des Segments, RG = rechte Grenze des Segments, LK = Zeiger auf die linke unmittelbare Konstituente, RK = Zeiger auf die rechte unmittelbare Konstituente. (Zu den fettgedruckten Einträgen siehe unter *Herstellung der Strukturbeschreibung.)*

Abb. 32.10: Bottom-up Parser nach Cocke, Younger und Kasami

NEU die rechte Grenze gleich der laufenden Position und die linke Grenze gleich der laufenden Position minus 1. Die Spalten für die unmittelbaren Konstituenten bleiben leer.

(A-2) Reduktion: Suche die nächste Produktion, in der das anstehende Symbol (d. i. die Kategorie in der Zeile LFD) mit der rechten unmittelbaren Konstituente übereinstimmt. Ist keine (mehr) vorhanden, so weiter (A-3). Suche im Arbeitsbereich eine Zeile ALT, deren Kategorie mit der linken Konstituente in der Produktion übereinstimmt und deren rechte Grenze gleich der linken Grenze in der Zeile LFD ist. (Die gesuchten Zeilen befinden sich in einem Teilbereich des Arbeitsbereichs mit den entsprechenden rechten Grenzen.) Falls eine solche vorhanden ist, so schreibe die Kategorie der Produktion in eine neue Zeile (erhöhe dabei NEU). Übernehme als linke Grenze in der Zeile NEU die linke Grenze der Konstituente in ALT und setze die rechte Grenze in der Zeile NEU auf den Wert der laufenden Position. Trage als linke Konstituente in der Zeile NEU einen Zeiger auf die Zeile ALT ein und als rechte Konstituente einen Zeiger auf die Zeile LFD. Weiter (A-2).

(A-3) Abbruchbedingung: Erhöhe die laufende Zeile LFD um 1. Übertrifft der Wert nicht den der bisher letzten Zeile im Arbeitsbereich (d. i. Zeile NEU), so weiter (A-2). Sonst rücke die laufende Position um eins vor. Ist damit das Eingabeende überschritten, so weiter (A-4), sonst weiter (A-1).

(A-4) Erfolgsprüfung: Befinden sich im Arbeitsbereich eine oder mehrere Zeilen mit linker Grenze gleich 0 und rechter Grenze gleich der Zahl der Eingabeelemente, so akzeptiere die Eingabe und beende die Prozedur. Sonst weise die Eingabe zurück und beende die Prozedur.

Anmerkung: Die Zusammenfassung mit den nach links hin anschließenden Segmenten (die je in einem bestimmten Teilbereich zu finden sind) ist die gleiche, wie beim Completor-Schritt des Earley-Algorithmus. Da eine Verbindung bis zum Satzanfang nicht gefordert wird, kommt es zu Zwischenergebnissen, die der Earley-Algorithmus vermeidet (z. B. (10) 'antworten auf disketten' als Verbalphrase). Dafür erzeugt der Earley-Algorithmus viele überflüssige Prädiktionen, die bei bottom-up Analyse mit rechtem Aufhänger nicht vorkommen.

Analysematrix. Im Originalalgorithmus von Cocke wird eine Matrix benutzt, deren Zeilen linke Grenzen ($0 \leq LG \leq n-1$) und deren Spalten rechte Grenzen ($1 \leq RG \leq n$) darstellen. Die Felder repräsentieren somit die möglichen Segmente verschiedener Länge. Ist ein Segment eine wohlgeformte Konstituente, wird deren Kategorie in das Feld eingetragen. (Vgl. Aho/Ullmann 1972, 314 ff.; Mayer 1978, 98 ff.)

Überträgt man die Kategorien aus dem Arbeitsbereich in eine solche Matrix, entsteht das Folgende:

LG \ RG	1	2	3	4	5
0	Nu		S		S, S
1		Vt	Vi		Vi, Vi
2			Vi, Nu		Vi, Nu
3				präp	PP
4					Nu

In ein Feld (i, j) der Matrix ist die Kategorie einer Produktion einzutragen, wenn die Kategorie der ersten unmittelbaren Konstituente in derselben Zeile (i, k) und die Kategorie der zweiten unmittelbaren Konstituente in derselben Spalte (k, j) steht und die Spaltennummer k der ersten Konstituente mit der Zeilennummer k der zweiten Konstituente übereinstimmen. Die Steuerung Breite-zuerst des klassischen Cocke-Algorithmus bedeutet, daß in der Diagonalen der Matrix von links oben nach rechts unten vorgegangen wird, während unser nach dem Prinzip Tiefe zuerst arbeitender Algorithmus die Felder von links nach rechts und von unten nach oben zu füllen versucht.

Herstellung der Strukturbeschreibung. Anhand der explizit gespeicherten linken und rechten unmittelbaren Konstituenten für jedes Segment läßt sich, beginnend mit der Kategorie des (bzw. eines) Segments mit linker Grenze gleich 0 und rechter Grenze gleich Anzahl der Eingabeelemente unmittelbar ein Phrasenstrukturbaum konstruieren. Natürlich kann ein solcher Strukturbaum auch

schon während der Analyse aufgebaut werden. (Vgl. dazu 2.2.2.)

Man kann aber auch einen *Dependenzbaum als Ergebnis herstellen (Hellwig, hier erstmals veröffentlicht). Jede Regel einer Grammatik in Chomsky-Normalform stellt nämlich eine syntagmatische Relation zwischen zwei Elementen dar, von denen unter Abhängigkeitsgesichtspunkt eines das dominierende, das andere das dependente Element in der Relation sein muß. In G4 (Abb. 32.10) wurde das dominierende Element (head constituent) durch Fettdruck hervorgehoben, im Arbeitsbereich wurden analog die Zeiger auf die dominierenden Konstituenten fett gedruckt. Mit Hilfe der Zeiger konstruiert man eine Liste wie folgt: Man beginnt mit der (bzw. einer) Zeile, mit der die Eingabe erkannt worden ist. Man ersetzt den Zeiger auf diese Liste durch den Zeiger auf die dominierende Konstituente in derselben Zeile und ordnet ihr den Zeiger auf die andere Konstituente unter. Für jeden der so ermittelten Zeiger in der Liste verfährt man ebenso, bis man überall auf leere rechte und linke Konstituenten gestoßen ist. Ersetzt man schließlich die lexikalischen Kategorien durch Lexeme, erhält man die gesuchte Dependenzstruktur. Für die beiden Ergebnisse im Arbeitsbereich oben, läuft das Verfahren folgendermaßen ab:

1. (14: S)
   (12: V_i (1: N_u))
   (5: V_i (1: N_u) ((9: PP))
   (2: V_t (1: N_u) (4: N_u) (9: PP))
   (2: V_t (1: N_u) (4: N_u) (7: präp (8: N_u)))
   (2: verarbeiten
   (1: computer)
   (4: antworten)
   (7: auf
   (8: disketten))
2. (15: S)
   (13: V_i (1: N_u))
   (2: V_t (1: N_u) (11: N_u))
   (2: V_t (1: N_u) (4: N_u (9: PP)))
   (2: V_t (1: N_u) (4: N_u (7: präp (8: N_u))))
   (2: verarbeiten
   (1: computer)
   (4: antworten
   (7: auf
   (8: disketten))))

An den Zeigern erkennt man deutlich, wie dieselben Teilergebnisse in den unterschiedlichen Analysen verwendet worden sind.

2.2.2. Verallgemeinerter Bottom-up Parser mit Teilergebnistabelle

Prinzip. Der positionsorientierte bottom-up Algorithmus mit Teilergebnistabelle läßt sich ausbauen, so daß die Grammatik Produktionen mit beliebig vielen unmittelbaren Konstituenten enthalten darf. Eine Produktion mit einer einzigen unmittelbaren Konstituente (Einer-Regel) führt dazu, daß dasselbe Segment eine weitere Kategorie zugeordnet bekommt. Produktionen mit mehreren Konstituenten werden mit der Teilergebnistabelle abgeglichen, indem eine entsprechende Folge von angrenzenden Segmenten auf der Grundlage der rechten und linken Grenzen gesucht wird. Der Algorithmus erlaubt ein kontinuierliches Vorrücken im Text, so daß der Parser parallel zur Eingabe (on-line) arbeiten kann. Interpunktionszeichen, die die selbständigen Einheiten im Text markieren, werden wie lexikalische Elemente behandelt. Beliebige syntaktische Kategorien können als mögliche selbständige Einheiten ausgezeichnet werden. Der zu parsende Text braucht also nicht nur aus Sätzen zu bestehen. In der folgenden Version enthält der Arbeitsbereich partielle Strukturbeschreibungen in Form von Listen. Listen als Teilergebnisse ermöglichen eine mehr als lokale Unifikation von komplexen grammatischen Kategorien. (Hays 1967, 123 ff.; Kuno 1976, 178 ff.)

Voraussetzungen:

(V-1) ein Lexikon, mit dessen Hilfe den Elementen der Eingabe lexikalische grammatische Kategorien zugewiesen werden.

(V-2) eine Variable, die die laufende Position in der Eingabe bezeichnet und zu Beginn den Wert 0 hat.

(V-3) die Menge der Produktionen einer kontextfreien Phrasenstrukturgrammatik ohne Tilgungen, erweitert um (beliebige) Produktionen der Form '$\Phi \rightarrow$ u', wobei 'u' für ein Symbol oder eine Kette von Symbolen der Grammatik steht und 'Φ' eine zusätzliche Kategorie ist, die die selbständigen Einheiten, aus denen ein Text besteht, bezeichnet. Für die Abarbeitung der Produktionen wird eine Laufvariable UK über den unmittelbaren Konstituenten benötigt.

(V-4) ein Arbeitsbereich, der eine Folge von Listen (bzw. Zeiger auf Listen) aufnehmen kann, sowie zu jeder Liste die linke und rechte Grenze des zugehöri-

GRAMMATIK G5 (schließt G3 ein, siehe Abb. 32.8)

Regeln:

(R-1)	Φ	→	S #
(R-2)	Φ	→	NP #
(R-3)	S	→	NP VP
(R-4)	VP	→	vi
(R-5)	VP	→	vt NP
(R-6)	VP	→	VP PP
(R-7)	NP	→	n
(R-8)	NP	→	adj_{ind} n
(R-9)	NP	→	det adj_{def} n
(R-10)	NP	→	NP PP
(R-11)	PP	→	präp NP

Lexikon:

#	=	{., ?, !}
vi	=	{fragen, antworten}
vt	=	{erzeugen}
det	=	{die, keine}
adj_{ind}	=	{natürlichsprachige}
adj_{def}	=	{passenden}
n	=	{benutzer, computer, systeme, fragen, antworten}
präp	=	{auf}

EINGABEBEREICH:

Eingabe:	natürlichsprachige systeme. benutzer fragen. computer erzeugen die passenden antworten.											
Lexikon:	adj_{ind}	n	#	n	vi/n	#	n	vt	det	adj_{def}	vi/n	#
Position:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)

ARBEITSBEREICH:

A:	LG:	RG:	Teilstruktur:	Erklärung:
(1)	0	1	adj_{ind}	Lexikon
(2)	1	2	n	Lexikon
(3)	1	2	NP (n)	R-7: (2)
(4)	0	2	NP (adj_{ind} n)	R-8: (2) + (1)
(5)	2	3	#	Lexikon
(6)	0	3	**Φ (NP (adj_{ind} n) #)**	R-2: (5) + (4)
(7)	3	4	n	Lexikon
(8)	3	4	NP (n)	R-7: (7)
(9)	4	5	vi	Lexikon
(10)	4	5	n	Lexikon
(11)	4	5	VP (vi)	R-4: (9)
(12)	4	5	NP (n)	R-7: (10)
(13)	3	5	S (NP (n) VP (vi))	R-3: (11) + (8)
(14)	5	6	#	Lexikon
(15)	3	6	**Φ (S (NP (n) VP (vi)) #)**	R-1: (14) + (13)
(16)	6	7	n	Lexikon
(17)	6	7	NP (n)	R-7: (16)
(18)	7	8	vt	Lexikon
(19)	8	9	det	Lexikon
(20)	9	10	adjdef	Lexikon
(21)	10	11	vi	Lexikon
(22)	10	11	n	Lexikon
(23)	10	11	VP (vi)	R-4: (21)
(24)	8	11	NP (det adj_{def} n)	R-9: (22) + (20) + (19)
(25)	7	11	VP (vt NP (det adj_{def} n))	R-5: (24) + (18)
(26)	6	11	S (NP (n) VP (vt NP (det adj_{def} n)))	R-3: (25) + (17)
(27)	11	12	#	Lexikon
(28)	6	12	**Φ (S (NP (n) VP (vt NP (det adj_{def} n))) #)**	R-1: (27) + (26)

Legende: P = laufende Position in der Eingabe, A = Numerierung der Teilergebnisse im Arbeitsbereich, über die die Variablen LFD und NEU laufen, LG = linke Grenze, RG = rechte Grenze.

Abb. 32.11: Verallgemeinerter Bottom-up Parser mit Teilergebnistabelle

gen Segments in der Eingabe, außerdem zwei Laufvariablen LFD und NEU über diesem Bereich sowie eine Variable ALT. Zu Beginn ist der Arbeitsbereich leer: LFD hat den Anfangswert 1 und NEU den Anfangswert 0.

Algorithmus:

(A-1) Einlesen und Lexikon: Erhöhe die laufende Position um 1 und lese das nächste Element in der Eingabe ein. (Ist die Eingabe erschöpft, beende die Prozedur.) Entnehme dem Lexikon die lexikalische(n) Kategorie(n) des eingelesenen Elements und schreibe sie als Liste(n) in den Arbeitsbereich (erhöhe dabei NEU); setze die linke Grenze in der Zeile NEU gleich der laufenden Position minus 1, und die rechte Grenze gleich der laufenden Position. Weiter (A-2).

(A-2) Grammatik: Suche eine (weitere) Produktion der Grammatik, deren äußerst rechte Konstituente mit der Kategorie in der Zeile LFD des Arbeitsbereichs übereinstimmt. Ist keine solche Produktion (mehr) vorhanden, so weiter (A-6). Bilde eine Liste mit der Kategorie der Produktion als einzigem Element. Setze die Laufvariable UK gleich der Anzahl der Konstituenten in der Produktion. Dadurch wird die äußerst rechte unmittelbare Konstituente in der Produktion zur aktuellen gemacht. Setze die Variable ALT mit der laufenden Zeile LFD gleich. Weiter (A-3).

(A-3) Reduktion: Unifiziere die Liste in der Zeile ALT mit der Konstituente UK in der Produktion (vgl. den Scanner-Schritt mit Unifikation für den Earley-Algorithmus oben unter 2.1.1., Herstellung einer Strukturbeschreibung). Scheitert die Unifikation, so weiter (A-2). Sonst ordne das Ergebnis als Teilliste der Liste mit der Kategorie der Produktion unter, wobei die grammatischen Merkmale in beiden nach Vorschrift der Produktion ebenfalls in Übereinstimmung zu bringen sind. Ist UK gleich 1, so weiter (A-4). Sonst reduziere den Wert von UK um 1. Suche im Arbeitsbereich eine Zeile, in der die rechte Grenze gleich der linken Grenze in der Zeile ALT ist. Wird keine gefunden, so weiter (A-2). Sonst mache diese Zeile zur neuen Zeile ALT und weiter (A-3).

(A-4) Speicherung: Speichere die neue Liste in einer neuen Zeile im Arbeitsbereich (erhöhe dabei NEU); übernehme den Wert für die linke Grenze aus der Zeile ALT und den Wert für die rechte Grenze aus der Zeile LFD. Ist die dominierende Kategorie der neuen Liste das Symbol Φ, so weiter (A-5). Sonst weiter (A-2).

(A-5) Ausgabe: Das Segment zur laufenden Zeile LFD ist als selbständige Texteinheit analysiert worden. ALT sei die linke Grenze der Zeile LFD. Ist ALT gleich 0 oder befindet sich in der Zeile ALT ebenfalls eine Liste mit dem Symbol Φ als dominierender Kategorie (d. h. ist das vorangehende Textstück ebenfalls als selbständig erkannt gewesen), so schreibe die Liste unter (Φ) in die Ausgabedatei. Ist dies nicht der Fall, erzeuge eine Fehlermeldung und schreibe die Liste unter (Φ) mit einer Warnung in die Ausgabedatei. Weiter (A-2).

(A-6) Schleifenbedingung: Ist LFD gleich NEU, so sind alle bisherigen Listen im Arbeitsbereich überprüft worden. Dann weiter (A-1). Sonst erhöhe LFD um 1 und weiter (A-2).

Anmerkung. Die Einteilung des Arbeitsbereiches zeigt das Vorrücken des Parsers um jeweils ein Eingabeelement. Der Parser kann, ebenso wie der Earley Parser, on-line betrieben werden. Die fettgedruckten Zeilen enthalten die Strukturbeschreibungen der selbständigen Einheiten des Textes. Sie bilden die Ausgabe. Bei einer realistischen Implementierung müßten komplexe Kategorien benutzt werden, so daß die Kongruenzen von Numerus, Kasus, Person, definiter und indefiniter Nominalphrase und Adjektivflexion richtig berechnet werden (vgl. dazu Brockhaus 1971).

3. Parsing mit Mustern und Übergangsnetzwerken

Eine Methode, um Objekte zu erkennen, ist ihr Vergleich mit vorgegebenen Mustern. Der Grundgedanke des Parsings mit *Mustern ist der des gleichzeitigen Fortschreitens innerhalb zweier Symbolsequenzen, dem Muster und der Eingabe, wobei bestimmte Bedin-

gungen festgelegt sind, unter denen ein Muster zu einer Eingabe paßt. Graphisch werden Muster häufig als Netze repräsentiert. Die Knoten repräsentieren Zustände des Erkenners bei der Abarbeitung des Musters, die Kanten sind mit den Symbolen etikettiert, die angeben, unter welcher Voraussetzung von einem Zustand zum nächsten übergegangen werden darf. Das *Erkennen eines Ausdrucks der Objektsprache erscheint als ein bestimmter Pfad durch das Netz. Bei der unmittelbar zeichenkettenorientierten Mustererkennung stehen die Symbole in den Mustern in direkter Analogie zu den Symbolen in der Eingabe. Prozedurale Netzwerke gleichen dagegen Flußdiagrammen; sie sind Muster für Programmabläufe, die zum Erkennen der Zeichenketten in der Eingabe führen.

3.1. Zeichenkettenorientierte Mustererkennung

Zur Formulierung von Mustern wird eine Metasprache definiert, deren Syntax Bedingungen für die Art der Symbole der Objektsprache sowie Notationsvereinfachungen für die Wiederholung von Zeichen, für optionale und für alternative Zeichenketten umfaßt. Muster können Namen von Mustern enthalten und auf diese Weise rekursiv gemacht werden. Die Aufeinanderfolge der Symbole in den Mustern entspricht der Konkatenation von Segmenten in der Eingabe. (Winograd 1983, 36ff.)

3.1.1. Erkenner für reguläre Ausdrücke

Prinzip: Zur Formulierung der Muster wird eine *reguläre Sprache definiert. Die in dieser Sprache gebildeten regulären Ausdrücke lassen sich von einem endlichen erkennenden Automaten den Ausdrücken der Objektsprache zuordnen. Dazu werden sie zunächst in ein nicht-deterministisches finites Übergangsnetzwerk (finite state transition network, FTN) umgeformt. Die Knoten in diesem Netz entsprechen den Grenzen zwischen Zeichenketten und bilden zugleich die Zustände des Automaten. Die Kanten symbolisieren die Übergänge von einer Position zur nächsten, wenn eine bestimmte Zeichenkette in der Eingabe vorliegt. Ein nicht-deterministisches Übergangsnetzwerk kann entweder mit Rücksetzen oder parallel abgearbeitet werden. In letzterem Fall wird in jedem Schritt die Menge der mit dem nächsten Eingabesymbol erreichbaren Zustände aus der Menge der bis dahin erreichten Zustände ermittelt. Am Schluß muß diese Menge einen Endzustand enthalten. (Vgl. Aho/Sethi/Ullman 1986, 130ff.; Winograd 1983, 61ff.) Aus einem nicht-deterministischen Übergangsnetzwerk kann ein deterministisches Übergangsnetz konstruiert werden, indem die Mengen von Zuständen im ersten Netz, die bei Vorliegen der verschiedenen Zeichenketten erreichbar sind, zu den Zuständen (Knoten) des zweiten Netzes gemacht werden. In diesem Fall führt jeder Schritt von einem Zustand aus über das nächste Symbol in der Eingabe zum nächsten Zustand oder zum Fehlschlagen der Analyse. Netze können als *Listen mit Zeigern oder in Form von Übergangstabellen gespeichert werden. Die Zeilen der Tabelle sind mit Ausgangszuständen und die Spalten mit Symbolen für elementare Zeichenketten indiziert. Die Felder der Tabelle enthalten die Zielzustände, zu denen bei Vorliegen der betreffenden Zeichenkette in der Eingabe übergegangen werden kann.

Sprache für reguläre Ausdrücke. Eine Sprache M zur Formulierung von Mustern für die Sprache L kann beispielsweise wie folgt definiert sein. (Vgl. Aho/Sethi/Ullman 1986, 94ff; ähnliche Konventionen gelten für die Funktion LEX des Betriebssystems UNIX.)

(i) 'ε' ist ein Ausdruck von M und bezeichnet ein Vorkommen der leeren Zeichenkette.

(ii) Ist a ein Element des Vokabulars von L, so ist a ein Ausdruck von M und bezeichnet ein Vorkommen des Elements a aus L.

(iii) Ist K eine Teilmenge des Vokabulars von L, so ist K ein Ausdruck von M und bezeichnet das Vorkommen eines beliebigen Elements der Kategorie K in L.

(iv) Sind r und s Ausdrücke von M, so ist rs ein Ausdruck von M und bezeichnet die Konkatenation der von r und s bezeichneten Zeichenketten in L.

(v) Sind r und s Ausdrücke von M, so ist r|s ein Ausdruck von M und bezeichnet das alternative Vorkommen der von r und s bezeichneten Zeichenketten in L.

(vi) Ist r ein Ausdruck von M, so ist r^+ ein Ausdruck von M und bezeichnet die Konkatenation aus einem oder mehreren Vorkommen der von r bezeichneten Zeichenkette in L.

(vii) Ist r ein Ausdruck von M, so ist r* ein Ausdruck von M und bezeichnet die Konkatenation der Zeichenketten aus null oder beliebig vielen Vorkommen der von r

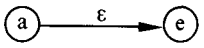
bezeichneten Zeichenkette in L, d. h. $r^* = r^+|\varepsilon$.

(viii) Ist r ein Ausdruck von M, so ist r? ein Ausdruck von M und bezeichnet ein optionales Vorkommen der von r bezeichneten Zeichenkette in L, d. h. $r? = r|\varepsilon$.

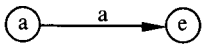
(ix) Die Klammerpaare '(',')' und '{','}' werden dazu benutzt, Ausdrücke von M hinsichtlich der Konkatenation und Alternation zu gruppieren. Ist r ein Ausdruck von M, so sind auch (r) und {r} Ausdrücke von M.

Konstruktion eines nicht-deterministischen Übergangsnetz zu regulären Ausdrücken. Zuerst werden den elementaren Ausdrücken in einem vorliegenden Muster nach (i) bis (iii) Übergangsnetzwerke zugeordnet; sodann werden nach (iv) bis (ix) die Netzwerke der Teilausdrücke miteinander kombiniert (vgl. Aho/Sethi/Ullman 1986, 124ff.). Der Anfangszustand des Erkenners bei der Ausarbeitung eines (Teil)ausdrucks sei a; der Endzustand sei e; ein Zwischenzustand sei Z_i. Zu jedem (Teil)ausdruck gibt es ein Netz mit genau einem Anfangs- und einem Endzustand.

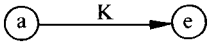
(i) Zum Ausdruck 'ε' konstruiere das Netzwerk



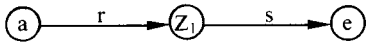
(ii) Zum Ausdruck a (für ein Element aus dem Vokabular von L) konstruiere das Netzwerk



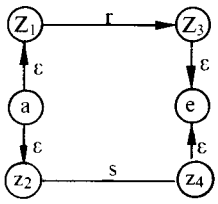
(iii) Zum Ausdruck K (für ein Element aus der Teilmenge K des Vokabulars von L) konstruiere das Netzwerk



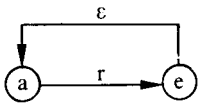
(iv) Zum Ausdruck rs konstruiere das Netzwerk



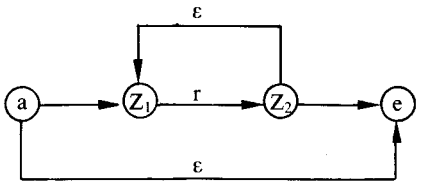
(v) Zum Ausdruck r|s konstruiere das Netzwerk



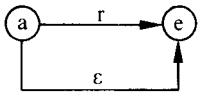
(vi) Zum Ausdruck r^+ konstruiere das Netzwerk



(vii) Zum Ausdruck r^* konstruiere das Netzwerk



(viii) Zum Ausdruck r? konstruiere das Netzwerk



(ix) Zu Ausdrücken, die von Klammerpaaren '(', ')' und '{', '}' umschlossen sind, konstruiere ein vollständiges Netzwerk, bevor dieses mit den Netzwerken zu den angrenzenden Ausdrücke kombiniert wird. Eine mehrfache Klammerung arbeite von innen nach außen ab.

Konstruktion einer Übergangstabelle für einen deterministischen endlichen Erkenner. Grundlage ist ein nicht-deterministisches Übergangsnetzwerk von der eben beschriebenen Art. Beginnend mit dem Anfangszustand werden alle Mengen von Zuständen gebildet, die vom gegebenen Zustand aus über ε-Kanten und/oder eine Kante mit einem elementaren Symbol erreicht werden können. (Dies entspricht der parallelen Abarbeitung des nicht-deterministischen Übergangsnetzwerkes.) Diese Zustandmengen bilden nun die Zustände in der Übergangstabelle für den deterministischen Erkenner. Als Ausgangszustände werden sie den Zeilen der Tabelle zugeordnet; als Zielzustände werden sie in die Spalten eingetragen, die den Elementen oder Kategorien des Vokabulars von L zugeordnet sind. Mit Hilfe der Tabelle lassen sich also deterministisch die Mengen aller möglichen Zustände bestimmen, in denen sich ein nicht-deterministischer Erkenner befinden kann, nachdem das nächste Eingabesymbol gelesen worden ist. (Vgl. Aho/Sethi/Ullman 1986,

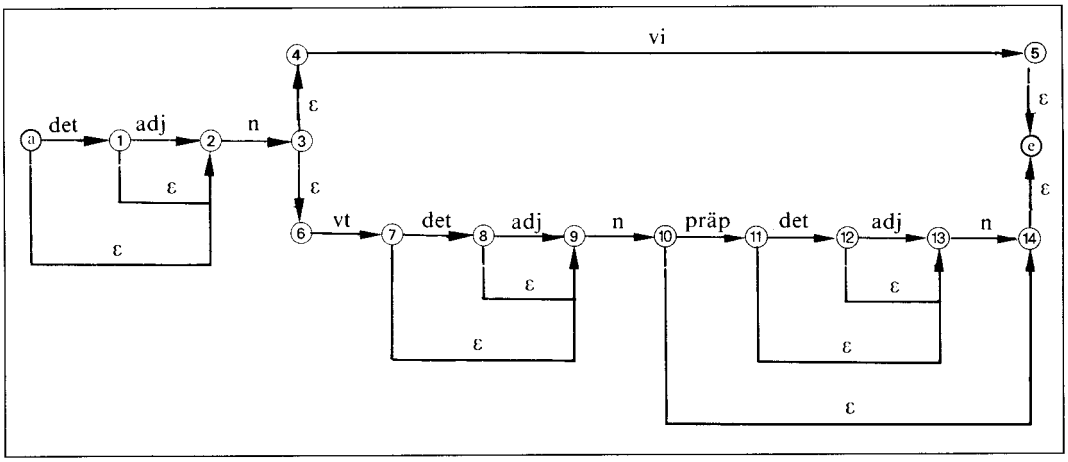
GRAMMATIK G6 (äquivalent zu G1, vgl. Abb. 32.1)

Muster:

(det adj?)? n {vi | vt (det adj?)? n (präp (det adj?) n)?}

Lexikon: wie G1

ÜBERGANGSNETZWERK:



ÜBERGANGSTABELLE FÜR EINEN DETERMINISTISCHEN ERKENNER:

Zustand:	det	adj	n	vi	vt	präp
a/2	1/2	—	3/4/6	—	—	—
1/2	—	2	3/4/6	—	—	—
2	—	—	3/4/6	—	—	—
3/4/6	—	—	—	5/e	7/9	—
5/e	—	—	—	—	—	—
7/9	8/9	—	10/14/e	—	—	—
8/9	—	9	10/14/e	—	—	—
9	—	—	10/14/e	—	—	—
10/14/e	—	—	—	—	—	11/13
11/13	12/13	—	14/e	—	—	—
12/13	—	13	14/e	—	—	—
13	—	—	14/e	—	—	—
14/e	—	—	—	—	—	—

Legende: Zur besseren Einsicht sind die Namen der Zustände in der Tabelle aus den Zuständen (Knoten) des Übergangsnetzwerkes zusammengesetzt worden. Normalerweise würden die Zustände in der Tabelle neu durchnumeriert werden. Mögliche Endzustände sind fettgedruckt.

EINGABEBEREICH:

Eingabe:	computer	erzeugen	antworten	nach	regeln
Lexikon:	n	vt	vi/n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

ENDLICHER ERKENNER:

Ausgangszustand:	**Position:**	**Eingabesymbol:**	**Zielzustand:**
a/2	1	n	3/4/6
3/4/6	2	vt	7/9
7/9	3	vi	—
7/9	3	n	10/14/e
10/14/e	4	präp	11/13
11/13	5	n	14/e
14/e	6	—	—

Abb. 32.12: Erkenner mit Finitem Übergangsnetzwerk (FTN)

117f., 132f.) Diejenigen Zustandsmengen, die den Endzustand des nicht-deterministischen Übergangsnetzwerkes einschließen, werden in der Tabelle als mögliche Endzustände des deterministischen Erkenners ausgezeichnet.

Voraussetzungen:

(V-1) eine Sprache zur Formulierung von Mustern in Form von regulären Ausdrücken.

(V-1) eine Grammatik der Objektsprache, die extern als Muster formuliert ist und intern die Form einer Übergangstabelle für einen deterministischen endlichen Automaten hat, deren Zeilen Ausgangszustände, deren Spalten lexikalische Kategorien der Eingabesprache und deren Einträge Zielzustände sind. Mögliche Endzustände sind besonders ausgezeichnet.

(V-2) ein Lexikon, mit dessen Hilfe den lexikalischen Elementen der Objektsprache Kategorien zugeordnet werden.

(V-3) eine Variable für die aktuellen Zustände des Erkenners, die zu Beginn den Anfangszustand der Übergangstabelle als Wert hat.

(V-4) eine Variable, die die laufende Position in der Eingabe bezeichnet und zu Beginn den Wert 1 hat.

Algorithmus:

(A-1) Übergang: Steht in der Spalte der Übergangstabelle, die dem Wortlaut oder der Kategorie des nächsten Elements in der Eingabe entspricht, ein Zustand, zu dem vom aktuellen Zustand aus übergegangen werden kann, so mache diesen Zustand zum aktuellen, erhöhe die laufende Position um 1 und weiter (A-2). Sonst weiter (A-3).

(A-2) Erfolgsprüfung: Ist die laufende Position nicht größer als die Zahl der Elemente in der Eingabe, so weiter (A-1). Ist der aktuelle Zustand ein Endzustand, so akzeptiere die Eingabe und beende die Prozedur.

(A-3) Mißerfolg: Gibt es eine alternative Kategorie zum Eingabeelement, so wähle diese und weiter (A-1). Sonst weise die Eingabe zurück und beende die Prozedur.

Anmerkung. Natürliche Sprachen als ganze gehören nicht zu den regulären, die mit Hilfe eines endlichen Automaten erkannt werden können, möglicherweise jedoch Ausschnitte davon, wie der Silbenbau, die Morphologie und die Wortbildung. Das Verfahren hat also durchaus Relevanz für beschränktere Aufgaben in der CL. In der Informatik wird das geschilderte Verfahren dazu benutzt, um die elementaren Zeichenketten herauszufinden, die anschließend an den Syntax-Parser übergeben werden. Man spricht daher auch von lexikalischer Analyse (lexical analysis). Das Verfahren ist auch zur partiellen syntaktischen Analyse von Texten geeignet (vgl. Rostek 1979). In diesem Fall werden diejenigen Zeichenketten ausgegeben, die zu einem Muster passen, d. h. an deren Anfang der Erkenner im Anfangszustand und an deren Ende er in einem der Endzustände ist. Der Rest des Textes wird übergangen. Wird immer so lange fortgefahren, bis ein Endzustand erreicht ist, von dem aus es keinen Übergang in einen neuen Zustand gibt, so werden die längsten Zeichenketten zum vorliegenden Muster gefunden (Prinzip des 'longest match'). Anschließend geht der Erkenner in einen neuen Anfangszustand über.

Herstellung der Strukturbeschreibung. Das Verfahren bietet als solches keine Handhabe, der erkannten Zeichenkette eine Struktur zuzuweisen. Es können jedoch Aktionen festgelegt werden, die auszuführen sind, wenn eine Instanz zu einem Muster gefunden worden ist. Diese Aktionen können auch in der Herstellung einer Repräsentation bestehen. (Aus Paaren von regulären Mustern und frei programmierten Aktionen bestehen z. B. die Spezifikationen für die Funktion LEX unter UNIX; vgl. Aho/Sethi/Ullman 1986, 107f.)

3.1.2. Parsing mit rekursiven Übergangsnetzwerken (RTN)

Prinzip. Die Kanten in einem rekursiven Übergangsnetzwerk (recursive transition network, RTN) können nicht nur, wie ein FTN, mit dem Symbol 'ε', einem lexikalischen Element oder einer lexikalischen Kategorie der Objektsprache indiziert sein, sondern auch mit dem Namen (bzw. den Anfangszustand) eines anderen (oder desselben) Netzwerkes. Der Übergang entlang einer solchen Kante ist erlaubt, wenn das angegebene Netzwerk vom Anfang bis zum Ende erfolgreich durchlaufen worden ist. Die Abarbeitung einer solchen Kante bedeutet also einen Sprung in ein Unternetzwerk; sie ist mit einem Unterpro-

gramm-Aufruf vergleichbar. Für gleiche Sprachausschnitte braucht nun nur noch einmal ein Netzwerk geschrieben zu werden. Linguistisch gesehen, entspricht ein Netzwerk einer nicht-terminalen Konstituente. Die Sprachen, die mit rekursiven Netzwerken erkannt werden können, sind dieselben wie diejenigen, welche von einer *kontextfreien Ersetzungsgrammatik generiert werden können. Kontext-freie Ersetzungsregeln bieten sich denn auch zur externen Sprachbeschreibung für RTN-Parser an. Sie lassen sich maschinell in rekursive Netzwerke überführen. Dabei werden alle Regeln mit demselben Symbol auf der linken Seite zu einem Netzwerk zusammengefaßt, das dieses Symbol als Anfangszustand enthält, und in dem für alle auf der rechten Seite der Regeln stehenden Symbolfolgen Pfade von diesem Anfangszustand in einen Endzustand erzeugt werden. Die im folgenden beschriebene nicht-deterministische Abarbeitung von rekursiven Übergangsnetzwerken läuft im Prinzip auf den in 1.1.1. beschriebenen top-down Parser mit Rücksetzen hinaus. Natürlich sind auch andere, z. B. parallele, Verfahren möglich. (Woods 1970, 591f. = 1976, 99f.; Winograd 1983, 196ff.; Aho/Sethi/Ullman 1986, 183ff.)

Voraussetzungen:

(V-1) bis (V-5) wie in 1.1.1. beim top-down Parser mit Rücksetzen.

(V-6) eine aus den Ersetzungsregeln einer kontext-freien Phrasenstrukturgrammatik erzeugte Menge von rekursiven Übergangsnetzwerken. Das Netzwerk, dessen Anfangszustand mit dem Startsymbol der Grammatik identisch ist, ist das ausgezeichnete.

(V-7) eine Variable für einen aktuellen Zustand im Netzwerk; zu Beginn ist dies der Anfangszustand des ausgezeichneten Netzwerkes.

(V-8) ein Arbeitsbereich, zur Speicherung von aktuellen Übergängen. Ein Übergang besteht aus dem Etikett einer Kante und einem Zielzustand.

(V-9) ein Rücksetzspeicher, in dem im Falle einer Alternativkante im Netzwerk die laufende Position im Eingabebereich, die laufende Zeile im Arbeitsbereich, der aktuelle Zustand, der alternative Übergang sowie der Inhalt des Stapelspeichers festgehalten werden.

(V-10) ein Stapelspeicher, der die Zielzustände der Kanten enthält, die zum Übergang in ein neues Netzwerk geführt haben und die erreicht sind, wenn das betreffende Netzwerk erfolgreich abgearbeitet wurde.

Algorithmus:

(A-1) Auswahl einer Kante: Ist der aktuelle Zustand der Endzustand in einem Netz, so weiter (A-5). Ist die laufende Position größer als die Zahl der Eingabeelemente, so weiter (A-7). Mache die erste vom aktuellen Zustand ausgehende Kante zum aktuellen Übergang. Speichere den aktuellen Übergang (auf einer neuen Zeile) im Arbeitsbereich. Weiter (A-2).

(A-2) Übergang: Gibt es im Netzwerk vom aktuellen Zustand aus eine weitere ausgehende Kante, so erzeuge einen Eintrag im Rücksetzspeicher mit der laufenden Position, der gegenwärtigen Zeile des Arbeitsbereichs, dem aktuellen Zustand sowie dem Etikett und dem Zielzustand der weiteren Kante.
Ist der aktuelle Übergang eine 'ε'-Kante, so mache den Zielzustand des aktuellen Übergangs zum neuen aktuellen Zustand und weiter (A-1). Ist das Etikett eine lexikalische Kategorie, so weiter (A-3). Ist das Etikett der Anfangszustand eines Netzwerks (d. i. eine nicht-terminale Kategorie), so weiter (A-4).

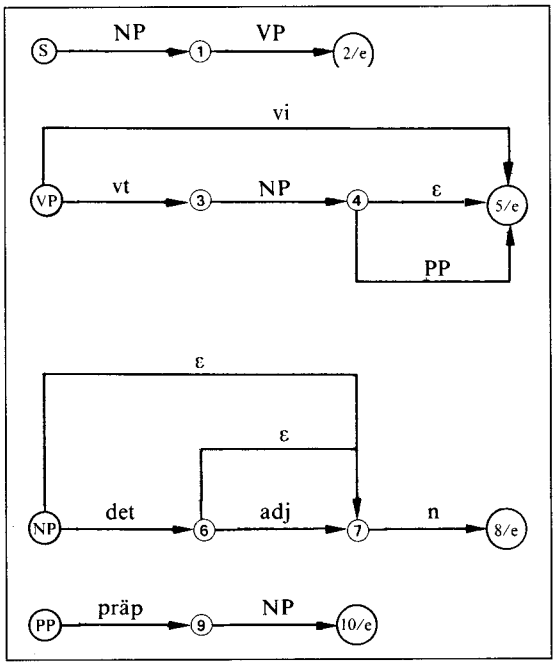
(A-3) Erkennen: Ist das Etikett des aktuellen Übergangs identisch mit der/einer Kategorie des Elements an der laufenden Position im Eingabebereich, so erhöhe die laufende Position um 1 und mache den Zielzustand des Übergangs zum neuen aktuellen Zustand und weiter (A-1). Andernfalls weiter (A-7).

(A-4) Sprung in ein Unternetzwerk: Setze den Zielzustand des aktuellen Übergangs auf den Stapelspeicher. Mache das Etikett des aktuellen Übergangs zum neuen aktuellen Zustand. Weiter (A-1).

(A-5) Rückkehr aus einem Unternetzwerk: Ist der Stapelspeicher leer, so weiter (A-6). Mache den letzten Eintrag im Stapelspeicher zum neuen aktuellen Zustand, entferne den Eintrag aus dem Stapelspeicher und weiter (A-1).

(A-6) Erfolgsprüfung: Ist die laufende Position gleich der Anzahl der Eingabeele-

GRAMMATIK G6 (rekursive Übergangsnetzwerke, äquivalent zu G1, vgl. Abb. 32.1)



EINGABEBEREICH:

Eingabe:	computer	erzeugen	antworten	nach	regeln
Lexikon:	n	vt	vi/n	präp	n
Position:	(1)	(2)	(3)	(4)	(5)

			ARBEITSBEREICH:			**RÜCKSETZSPEICHER:**						**STAPEL:**
P:	**B:**	**Z_a:**	**A:**	**K:**	**Z_z:**	**B:**	**P:**	**A:**	**Z_a:**	**K:**	**Z_z:**	**Z_z:**
1	0	S	(1)	NP	1							—
1	1	NP	(2)	ε	7	(1)	1	2	NP	det	6	1
1	1	7	(3)	n	8/e							1
2	1	8/e										—
2	1	1	(4)	VP	2/e							—
2	2	VP	(5)	vi	5/e	(2)	2	5	VP	vt	3	2/e
2	1	VP	(5)	vt	3							2/e
3	1	3	(6)	NP	4							2/e
3	2	NP	(7)	ε	7	(2)	3	7	NP	det	6	2/e, 4
3	2	7	(8)	n	8/e							2/e, 4
4	2	8/e										2/e, 4
4	3	4	(9)	ε	5/e	(3)	4	9	4	PP	5/e	2/e
4	3	5/e										—
4	2	4	(9)	PP	5/e							2/e
4	2	PP	(10)	präp	9							2/e, 5/e
5	2	9	(11)	NP	10/e							2/e, 5/e
5	3	NP	(12)	ε	7	(3)	5	12	NP	det	6	2/e, 5/e, 10/e
5	3	7	(13)	n	8/e							2/e, 5/e, 10/e
6	3	8/e										2/e, 5/e, 10/e
6	3	10/e										2/e, 5/e
6	3	5/e										2/e
6	3	2/e										—

Legende: P = laufende Position im Eingabebereich, A = Zeilen im Arbeitsbereich, B = aktueller Eintrag im Rücksetzspeicher, Z_a = aktueller Zustand, K = Kantenetikett, Z_z = Zielzustand.

Abb. 32.13: Parsing mit rekursiven Übergangsnetzwerken (RTN)

mente, so akzeptiere die Eingabe und beende die Prozedur. Sonst weiter (A-7).

(A-7) Rücksetzen: Ist der Rücksetzspeicher leer, so weise die Eingabe zurück und beende die Prozedur. Identifiziere die laufende Position, die laufende Zeile im Arbeitsbereich, den aktuellen Zustand sowie den Übergang in dieser Zeile mit den entsprechenden Werten im Rücksetzspeicher. Stelle den alten Zustand im Stapelspeicher wieder her. Entferne den Eintrag vom Rücksetzspeicher und weiter (A-2).

Anmerkung. Der Vorteil von rekursiven Netzwerken gegenüber einer Ableitung nach Ersetzungsregeln liegt darin, daß gleiche Abschnitte zu Beginn und am Ende einer Konstituentenfolge zusammengefaßt werden können. Anders als bei einer Regelgrammatik in *faktorisierter Normalform, resultieren daraus jedoch keine zusätzlichen Knoten im Phrasenstrukturbaum. Das Parsing mit RTNs ergibt also häufig flachere Bäume als die regelgeleiteten Verfahren. Nachteilig ist, daß komplexe Kategorien verschiedener Konstituenten nur schwer in Übereinstimmung gebracht werden können (*Unifikation), weil bei Abarbeitung einer Kante die Kategorien an den übrigen Kanten des Netzwerkes nicht zur Verfügung stehen. Um dieses Problem zu lösen, benötigt man Register wie in den im folgenden beschriebenen erweiterten Übergangsnetzwerken.

Herstellung der Strukturbeschreibung. Als Strukturbeschreibung wird ein Phrasenstrukturbaum erzeugt, dessen Knoten mit dem Namen des Anfangszustandes und den Etiketten der traversierten Kanten beschriftet werden, und dessen Unterordnungsverhältnisse sich aus der Rekursionsgeschichte der durchlaufenden Netzwerke ergeben.

3.2. Prozedurale Netzwerke

Man kann Übergangsnetzwerke statisch als Muster ansehen, die mit der Eingabe verglichen werden. Das ist aber etwas nicht sehr viel anderes als ein Vergleich der Eingabe mit Ableitungsgregeln. Man kann die Übergänge und die Unternetzwerke aber auch als *Prozeduren betrachten und als solche implementieren. Diese letztere Sicht, mit ihren Vor- und Nachteilen (siehe die Diskussion in 1.2. des Artikels 31) ist das besondere an prozeduralen Netzwerken. Die Netze sind hier im Prinzip Flußdiagramme für ein Programm zum Erkennen (oder Erzeugen) von Sprache. Sie schließen beliebige Tests und Aktionen, Unterprogrammaufrufe, die Übergabe von Parametern und die Berechnung von Werten für Parameter ein. Festlegungen für die Spezifikationen solcher Netzwerke haben eher den Status der Definition einer besonderen Programmiersprache als den eines Grammatikformalismus.

3.2.1. Erweiterte Übergangsnetzwerke (ATN)

Prinzip. Ausgangspunkt sind die RTN-Graphen wie in 3.1.2., mit dem Zeichen für die leere Zeichenkette, mit einer lexikalischen Kategorie oder mit dem Namen eines Unternetzwerks an den Kanten. Diese Netzwerke werden nun erweitert (daher 'augmented transition network', ATN), indem das Beschreiten einer Kante von beliebigen Bedingungen abhängig gemacht wird und indem damit beliebige Aktionen verknüpft werden. Ausschlaggebend ist dabei, daß Register eingerichtet werden können, in denen zu einem beliebigen Zeitpunkt Informationen abgelegt und zu einem anderen Zeitpunkt wieder entnommen werden können. Hat ein rekursives Netzwerk die Stärke eines Kellerautomaten und ist damit nur zur Analyse von kontextfreien Sprachen ausreichend, so entspricht ein erweitertes Netzwerk einer Turing-Maschine, d. h. es sind damit theoretisch alle aufzählbaren Sprachen erzeugbar und die Unterklasse der kontext-sensitiven Sprachen auch entscheidbar. Das Setzen und Abfragen von Registern ermöglicht es, kontextuelle Beziehungen zwischen den Elementen in der Eingabe zu berücksichtigen. Die Ablage von Information in Registern erlaubt zudem den Aufschub von Entscheidungen und vermindert so die Notwendigkeit des Rücksetzens. Strukturbeschreibungen werden durch eigene Aktionen erzeugt und können daher beliebiges Aussehen haben. Das Prinzip eines ATN-Parsers besteht darin, von einer aktuellen Konfiguration aus eine neue Konfiguration zu berechnen, indem versucht wird einer Kante zu folgen und dabei Tests und Aktionen durchzuführen. (Woods 1970 (deutsch 1976); Woods 1973; Bates 1978 (deutsch 1980); Billmeier 1979; Koch 1979; Lötscher 1979; Christaller/Metzing 1980; Winograd 1983, 195 ff.)

Definition einer ATN-Programmiersprache. Für eine bestimmte ATN-Implementierung muß vorab festgelegt sein, welche formalen Konstrukte ein erweitertes Übergangsnetzwerk enthalten darf, weil es sonst unmöglich wäre, die Übersicht über die Sprachstrukturen zu behalten, welche das ATN-Programm akzeptiert. Das folgende ist ein Beispiel einer solchen Festlegung. Zur Programmierung wird häufig (aber nicht immer) die Sprache LISP verwendet, in der die Evaluierung von Funktionen und die Kontrolle über die Rekursion bereits eingebaut sind. Für den Leser, der mit LISP nicht vertraut ist, ist die Notation jedoch verwirrend. Darum wird hier eine einfachere Darstellung gewählt. (Definitionen von ATNs in LISP findet man u. a. in Bates 1978 = 1980, mit einer kurzen Einführung in LISP; den LISP-Code für einen ATN-Interpreter enthält Christaller/Metzing 1980, Bd. 2, 173 ff.)

Im Folgenden bedeutet '→' „besteht aus"; '*' als Exponent besagt, daß das betreffende Element nicht oder auch mehrfach vorkommen kann; '|' trennt Alternativen; kursiv gesetzte Elemente sind im Rahmen der folgenden Syntax-Definition terminal, Näheres zu ihnen findet sich in der Beschreibung des Algorithmus; die großgeschriebenen Elemente sind Konstanten. Die Bedeutung der Konstrukte wird innerhalb des Algorithmus erläutert.

⟨Erweitertes Übergangsnetzwerk⟩ → ⟨Kantenmenge⟩*

⟨Kantenmenge⟩ → ⟨*Knoten*⟩ ⟨Kante⟩*

⟨Kante⟩ → ⟨WRD-Kante⟩ | ⟨CAT-Kante⟩ | ⟨JUMP-Kante⟩ | ⟨VIR-Kante⟩ | ⟨PUSH-Kante⟩ | ⟨POP-Kante

⟨WRD-Kante⟩ → WRD ⟨*lexikalisches Element*⟩ ⟨Test⟩* ⟨Aktion⟩* ⟨Übergang⟩

⟨CAT-Kante⟩ → CAT ⟨*lexikalische Kategorie*⟩ ⟨Test⟩* ⟨Aktion⟩* ⟨Übergang⟩

⟨JUMP-Kante⟩ → JUMP ⟨Übergang⟩ ⟨Test⟩* ⟨Aktion⟩*

⟨VIR-Kante⟩ → VIR ⟨*Konstituente*⟩ ⟨Test⟩* ⟨Aktion⟩* ⟨Übergang⟩

⟨PUSH-Kante⟩ → PUSH ⟨*Knoten*⟩ ⟨Test⟩* ⟨Prä-Aktion⟩*⟨Test⟩*⟨Aktion⟩*⟨Übergang⟩

⟨POP-Kante⟩ → POP ⟨Form⟩ ⟨Test⟩*

⟨Test⟩ → T | ⟨*Bedingung*⟩

⟨Prä-Aktion⟩ → ⟨Register-Weitergabe⟩

⟨Aktion⟩ → ⟨*Bedingung*⟩ ⟨Register-Eintrag⟩ | ⟨Register-Rückgabe⟩ | ⟨HOLD-Aktion⟩

⟨Register-Eintrag⟩ → SETR ⟨*Register*⟩ ⟨Form⟩ | ADDR ⟨*Register*⟩ ⟨Form⟩

⟨Register-Weitergabe⟩ → SENDR ⟨*Register*⟩ ⟨Form⟩

⟨Register-Rückgabe⟩ → LIFTR ⟨*Register*⟩ ⟨Form⟩

⟨HOLD-Aktion⟩ → HOLD ⟨*Konstituente*⟩ ⟨Form⟩

⟨Form⟩ → ⟨aktueller Eingabeausdruck⟩ | ⟨aktuelles Merkmal⟩ | ⟨Register-Inhalt⟩ | ⟨Konstruktionsanweisung⟩ | ⟨Form⟩ ∩ ⟨Form⟩

⟨aktueller Eingabeausdruck⟩ → *

⟨aktuelles Merkmal⟩ → GETF ⟨*Parameter*⟩ | „⟨*Wert*⟩"

⟨Register-Inhalt⟩ → GETR ⟨*Register*⟩

⟨Konstruktionsanweisung⟩ → BUILDQ ⟨*Rahmen*⟩ ⟨*Register*⟩*

⟨Übergang⟩ → TO ⟨*Knoten*⟩

Voraussetzungen:

(V-1) bis (V-4) wie beim top-down Parser mit Rücksetzen (siehe 1.1.1.).

(V-5) ein erweitertes Übergangsnetzwerk entsprechend obiger Definition; ein Knoten darin ist als Anfangsknoten ausgezeichnet.

(V-6) Variablen für den jeweils aktuellen Knoten und die aktuelle Kante beim Durchlaufen des Netzes.

(V-7) ein Stapelspeicher, in dem die aktuelle Konfiguration, bestehend aus dem aktuellen Knoten, der aktuellen Kante im Netzwerk und der Zustand der Registerliste, festgehalten wird, bevor bei der Abarbeitung einer PUSH-Kante die tiefere Rekursionsebene betreten wird.

(V-8) eine Liste von Registerlisten. Eine Registerliste ist eine Liste mit Registername-Registerwert Paaren. (Diese Organisation ist notwendig, um beliebige Register setzen zu können). Die Aktion SETR fügt ein neues Paar oben auf; dadurch sind die alten Name-Wert-Paare vor GETR verborgen, können aber beim Rücksetzen wieder sichtbar gemacht werden (für die Speicherung eines Zustandes braucht nur noch ein Pointer auf den aktuellen Beginn der Liste gesetzt werden). Auf jeder Rekursionsstufe ist eine andere Liste aus der Liste der Registerlisten aktuell. (Vgl. Woods 1973 b, 131 f.)

(V-9) eine HOLD-Liste, in der Information

über bisher bearbeitete Konstituenten abgelegt wird.

(V-10) ein Rücksetzspeicher, in dem im Falle einer Alternativkante die laufende Position in der Eingabe, der aktuelle Knoten und die alternative Kante im Netzwerk sowie der Zustand des Stapelspeichers, der Registerliste und der HOLD-Liste festgehalten werden.

Algorithmus:

(A) Abarbeitung des Übergangsnetzwerks:

(A-1) Erzeuge eine Anfangskonfiguration: die laufende Position in der Eingabe ist gleich 1; der aktuelle Knoten ist der Anfangsknoten; die aktuelle Kante ist die erste, welche den Anfangsknoten verläßt; die aktuelle Registerliste ist leer;

(A-2) 'Schnappschuß': Gibt es eine nächste Kante nach der aktuellen Kante, so erzeuge einen Eintrag im Rücksetzspeicher mit der laufenden Position, dem aktuellen Knoten, der nächsten Kante, dem Zustand der aktuellen Registerliste, dem Zustand der HOLD-Liste und dem Zustand des Stapelspeichers.
Bearbeite die aktuelle Kante je nach Kantenart wie folgt:
WRD-Kante: Stimmt das angegebene lexikalische Element mit dem an der laufenden Eingabeposition befindlichen überein, so tue das folgende; sonst weiter (A-3). Führe den angegebenen Test durch; schlägt er fehl, so weiter (A-3). Führe die angegebenen Aktionen durch. Erhöhe die laufende Position in der Eingabe um 1. Mache den TO-Knoten zum aktuellen Knoten und dessen erste Kante zur aktuellen Kante. Weiter (A-2).
CAT-Kante: Stimmt die angegebene Kategorie mit einer der lexikalischen Kategorien des Element an der laufenden Eingabeposition überein, so tue das folgende: sonst weiter (A-3). Führe den angegebenen Test durch; schlägt er fehl, so weiter (A-3). Führe die angegebenen Aktionen durch. Erhöhe die laufende Position in der Eingabe um 1. Mache den TO-Knoten zum aktuellen Knoten und dessen erste Kante zur aktuellen Kante. Weiter (A-2).
JUMP-Kante: Führe den angegebenen Test durch; schlägt er fehl, so weiter (A-3). Führe die angegebenen Aktionen durch. Ohne die Position in der Eingabe weiterzurücken, mache den TO-Knoten zum aktuellen Knoten und dessen erste Kante zur aktuellen Kante. Weiter (A-2).
VIR-Kante: Steht die angegebene Konstituente infolge einer HOLD-Aktion an einer früheren Kante auf der HOLD-Liste, so tue das folgende; sonst weiter (A-3). Führe den angegebenen Test durch; schlägt er fehl, so weiter (A-3). Führe die angegebenen Aktionen durch. Ohne die Position in der Eingabe weiterzurücken, mache den TO-Knoten zum aktuellen Knoten und dessen erste Kante zur aktuellen Kante. Weiter (A-2).
PUSH-Kante: Führe den angegebenen Test durch; schlägt er fehl, so weiter (A-3). Führe die angegebenen Prä-Aktion aus. Setze die aktuelle Konfiguration auf den Stapelspeicher. Erzeuge eine neue Konfiguration für die nächst tiefere Rekursionsstufe: mache den angegebenen Knoten zum aktuellen Knoten und die erste Kante, die ihn verläßt, zur aktuellen Kante; übernehme die laufende Position; beginne eine neue Registerliste. Weiter (A-2).
POP-Kante: Führe den angegebenen Test durch; schlägt er fehl, so weiter (A-3). Handelt es sich um die oberste Rekursionsstufe (d. h. der Stapelspeicher ist leer), so weiter (A-4). Sonst stelle die Konfiguration der nächst höheren Rekursionsstufe anhand des obersten Eintrags auf dem Stapelspeicher wieder her und entferne den Eintrag vom Stapel. Gibt die an der Kante angegebene Form sowie die erreichte laufende Position als Ergebnis des Unternetzwerks an die übergeordnete Rekursionsstufe zurück. Führe nun die Aktionen an der PUSH-Kante aus, die zum Aufruf des Unternetzwerkes geführt hatte. Mache den TO-Knoten der PUSH-Kante zum aktuellen Knoten und dessen erste Kante zur aktuellen Kante. Weiter (A-2).

(A-3) Gibt es einen Eintrag im Rücksetzspeicher, so mache die dortige Konfiguration zur aktuellen und setze auch den Stapelspeicher auf den angegebenen Zustand zurück; entferne den

Eintrag vom Rücksetzspeicher und weiter (A-2). Sonst melde Fehlschlag und halte an.

(A-4) Ist das Ende der Eingabe erreicht (d. h. die laufende Position ist um 1 größer als die Zahl der Wörter in der Eingabe), so ist die Analyse erfolgreich abgeschlossen. Gib das Ergebnis aus und halte an. Sonst weiter (A-3). (Sollen alle möglichen Lesungen einer Eingabe gefunden werden, so auch im Falle erfolgreicher Analyse weiter zu (A-3).)

(B) Durchführung von Tests:

Ein Test besteht darin, den Wahrheitswert für eine Bedingung zu ermitteln. Die Bedingung kann im Prinzip beliebige Prädikate und Argumente enthalten. Die im Beispiel benutzten

GRAMMATIK G7

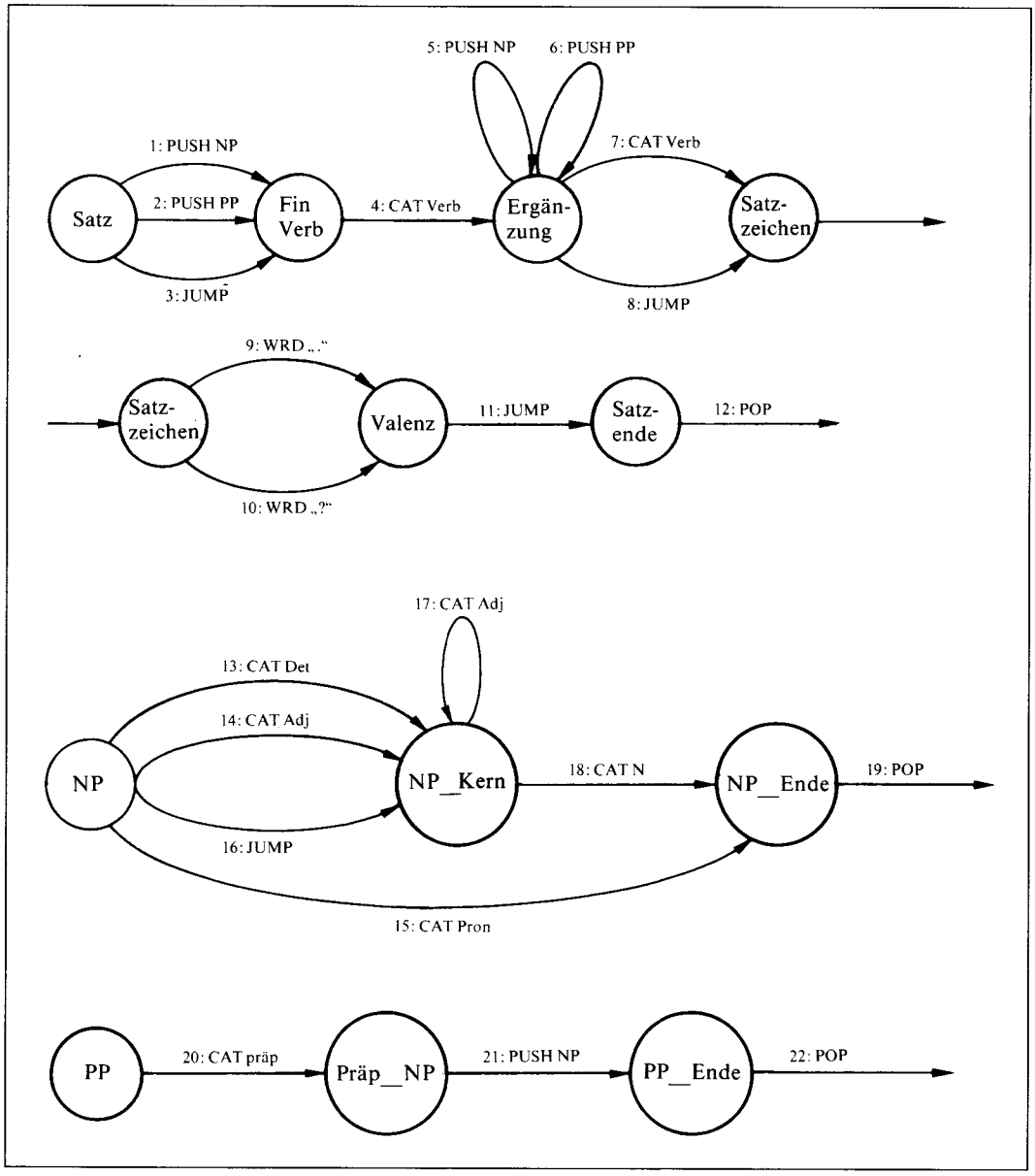


Abb. 32.14a: Erweiterte Übergangsnetzwerke (ATN)

ATN-PROGRAMM:
(Kursiv gesetzte Bezeichnungen sind zur besseren Verständlichkeit hinzugefügt. Funktionen, die einen Wert ergeben (also vor Benutzung evaluiert werden) sind in spitze Klammern eingeschlossen.)

Knoten **Satz**

Kante (1) PUSH NP
- *Aktion* wenn „nom" ε ⟨GETR Kasus⟩ dann ADDR Kongurenz ⟨GETR Person⟩ und ADDR Kongruenz ⟨GETR Numerus⟩ und SETR Subjekt ⟨BUILDQ (Subjekt: *)⟩
- *Aktion* wenn „gen" ε ⟨GETR Kasus⟩ dann SETR GenObjekt ⟨BUILDQ (GenObjekt: *)⟩
- *Aktion* wenn „dat" ε ⟨GETR Kasus⟩ dann SETR DatObjekt ⟨BUILDQ (DatObjekt: *)⟩
- *Aktion* wenn „akk" ε ⟨GETR Kasus⟩ dann SETR AkkObjekt ⟨BUILDQ (AkkObjekt: *)⟩
- *Aktion* wenn ⟨GETR WPron⟩ = „+" dann SETR Satz „W-frage:" sonst SETR Satz „aussage:"
- *Übergang* TO FinVerb

Kante (2) PUSH PP
- *Aktion* SETR PräpPhrase *
- *Aktion* wenn ⟨GETR WPron⟩ = „+" dann SETR Satz „W-frage:" sonst SETR Satz „aussage:"
- *Übergang* TO FinVerb

Kante (3) JUMP
- *Aktion* SETR Satz „E-frage:"
- *Übergang* TO FinVerb

Knoten **FinVerb**

Kante (4) CAT Verb
- *Test* ⟨GETF Form⟩ = „finit"
- *Test* wenn ⟨GETR Subjekt⟩ nicht = „F" dann ⟨GETF Person⟩ ∩ ⟨GETR Kongruenz⟩ nicht = „F" und ⟨GETF Numerus⟩ ∩ ⟨GETR Kongruenz⟩ nicht = „F"
- *Aktion* SETR Verb *
- *Aktion* SETR Tempus ⟨GETF Tempus⟩
- *Aktion* SETR Valenz ⟨GETF Valenz⟩
- *Aktion* wenn ⟨GETR Subjekt⟩ nicht = „F" dann SETR Numerus ⟨⟨GETF Numerus⟩ ∩ ⟨GETR Kongruenz⟩⟩ sonst SETR Kongruenz ⟨GETF Person⟩ und ADDR Kongruenz ⟨GETF Numerus⟩
- *Übergang* TO Ergänzung

Knoten **Ergänzung**

Kante (5) PUSH NP
- *Test* wenn „nom" ε ⟨GETR Kasus⟩ und ⟨GETR Subjekt⟩ = „F" dann ⟨GETF Person⟩ ∩ ⟨GETR Kongruenz⟩ nicht = „F" und ⟨GETF Numerus⟩ ∩ ⟨GETR Kongruenz⟩ nicht = „F"
- *Aktion* wenn „nom" ε ⟨GETR Kasus⟩ und ⟨GETR Subjekt⟩ = „F" dann SETR Subjekt ⟨BUILDQ (Subjekt: *)⟩
- *Aktion* wenn „gen" ε ⟨GETR Kasus⟩ dann SETR GenObjekt ⟨BUILDQ (GenObjekt: *)⟩
- *Aktion* wenn „dat" ε ⟨GETR Kasus⟩ dann SETR DatObjekt ⟨BUILDQ (DatObjekt: *)⟩
- *Aktion* wenn „akk" ε ⟨GETR Kasus⟩ dann SETR AkkObjekt ⟨BUILDQ (AkkObjekt: *)⟩
- *Übergang* TO Ergänzung

Kante (6) PUSH PP
- *Aktion* SETR PräpPhrase *
- *Übergang* TO Ergänzung

Kante (7) CAT Verb
- *Test* ⟨GETF Form⟩ = „partizip"
- *Test* ⟨GETR Valenz⟩ = „partizip"
- *Aktion* SETR Tempus ⟨GETR Verb⟩
- *Aktion* SETR Verb *
- *Aktion* SETR Valenz ⟨GETF Valenz⟩

Fortsetzung

Kante (8) JUMP
Test ⟨GETR Verb⟩ nicht = „F"
Übergang TO Satzzeichen

Knoten **Satzzeichen**
Kante (9) WRD „."
Test ⟨GETR Satz⟩ = „aussage"
Übergang TO Valenz
Kante (10) WRD „?"
Test ⟨GETR Satz⟩ = „W-frage" oder ⟨GETR Satz⟩ = „E-frage"
Übergang TO Valenz

Knoten **Valenz**
Kante (11) JUMP
Aktion wenn „subj" ε Valenz dann SETR Komplemente ⟨GETR Subjekt⟩
Aktion wenn „genobj" ε Valenz dann ADDR Komplemente ⟨GETR GenObjekt⟩
Aktion wenn „datobj" ε Valenz dann ADDR Komplemente ⟨GETR DatObjekt⟩
Aktion wenn „akkobj" ε Valenz dann ADDR Komplemente ⟨GETR AkkObjekt⟩
Aktion wenn „präpobj" ε Valenz und ⟨GETR Präposition⟩ ε Valenz
dann ADDR Komplemente ⟨BUILDQ (PräpObjekt: +) ⟨GETR PräpPhrase⟩⟩
sonst wenn ⟨GETR PräpPhrase⟩ nicht = „F"
dann ADDR Komplemente ⟨BUILDQ (Adverbiale: +) ⟨GETR PräpPhrase⟩⟩
Übergang TO Satzende

Knoten **Satzende**
Kante (12) POP
⟨BUILDQ (Illokution: + (Prädikat: + (Tempus: +) +))) Satz Verb Tempus Komplemente⟩

Knoten **NP**
Kante (13) CAT Det
Aktion SETR Kasus ⟨GETF Kasus⟩
Aktion SETR Genus ⟨GETF Genus⟩
Aktion SETR Numerus ⟨GETF Numerus⟩
Aktion SETR Flexion ⟨GETF Flexion⟩
Aktion SETR Artikel ⟨BUILDQ (Deter: *)⟩
Aktion LIFTR WPron ⟨GETF WPron⟩
Übergang TO NP-Kern
Kante (14) CAT Adj
Test ⟨GETF Flexion⟩ = „indef"
Aktion SETR Kasus ⟨GETF Kasus⟩
Aktion SETR Genus ⟨GETF Genus⟩
Aktion SETR Numerus ⟨GETF Numerus⟩
Aktion SETR Flexion ⟨GETF Flexion⟩
Aktion SETR Attribute ⟨BUILDQ (Attrib: *)⟩
Übergang TO NP-Kern
Kante (15) CAT Pron
Aktion SETR Kern ⟨BUILDQ (*)⟩
Aktion LIFTR Kasus ⟨GETF Kasus⟩
Aktion LIFTR Person ⟨GETF Person⟩
Aktion LIFTR Numerus ⟨GETF Numerus⟩
Aktion LIFTR WPron ⟨GETF WPron⟩
Übergang TO NP-Ende
Kante (16) JUMP
Übergang TO NP-Kern

Fortsetzung

Knoten **NP-Kern**
Kante (17) CAT Adj
Test ⟨GETR Kasus⟩ ∩ ⟨GETF Kasus⟩ nicht = „F"
Test ⟨GETR Genus⟩ ∩ ⟨GETF Genus⟩ nicht = „F"
Test ⟨GETR Numerus⟩ ∩ ⟨GETF Numerus⟩ nicht = „F"
Test ⟨GETR Flexion⟩ ∩ ⟨GETF Flexion⟩ nicht = „F"
Aktion SETR Kasus ⟨⟨GETR Kasus⟩ ∩ ⟨GETF Kasus⟩⟩
Aktion SETR Genus ⟨⟨GETR Genus⟩ ∩ ⟨SETF Genus⟩⟩
Aktion SETR Numerus ⟨⟨GETR Numerus⟩ ∩ ⟨GETF Numerus⟩⟩
Aktion SETR Flexion ⟨⟨GETR Flexion⟩ ∩ ⟨GETF Flexion⟩⟩
Aktion ADDR Attribute ⟨BUILDQ (Attrib: *)⟩
Übergang TO NP-Kern
Kante (18) CAT N
Test ⟨GETR Kasus⟩ ∩ ⟨GETF Kasus⟩ nicht = „F"
Test ⟨GETR Genus⟩ ∩ ⟨GETF Genus⟩ nicht = „F"
Test ⟨GETR Numerus⟩ ∩ ⟨GETF Numerus⟩ nicht = „F"
Aktion SETR Kern ⟨BUILDQ * (Numerus: +) Numerus⟩
Aktion LIFTR Kasus ⟨⟨GETR Kasus⟩ ∩ ⟨GETF Kasus⟩⟩
Aktion LIFTR Numerus ⟨⟨GETR Numerus⟩ ∩ ⟨GETF Numerus⟩⟩
Aktion LIFTR Person „dritte"
Übergang TO NP-Ende

Knoten **NP-Ende**
Kante (19) POP
⟨BUILDQ + + + Kern Artikel Attribute⟩

Knoten **PP**
Kante (20) CAT Präp
Aktion SETR Rektion ⟨GETF Kasus⟩
Aktion SETR Präposition *
Aktion LIFTR Präposition
Übergang TO Präp-NP

Knoten **Präp-NP**
Kante (21) PUSH NP
Aktion LIFTR WPron ⟨GETR WPron⟩
Übergang TO PP-Ende

Knoten **PP-Ende**
Kante (22) POP ⟨BUILDQ + (+) Präposition Nominalphrase⟩
Test ⟨GETR Rektion⟩ = ⟨GETR Kasus⟩

Abb. 32.14b: Ein ATN-Programm

LEXIKON:
(Der Wert, den das Lexikon der Variablen * für den aktuellen Eingabeausdruck zuweist, ist das Lexem bzw. die Grundform des Wortes.)

der	Det Kasus[nom] Genus[mas] Numerus[singular] Flexion[def]
der	Det Kasus[gen, dat] Genus[fem] Numerus[singular] Flexion[def]
der	Det Kasus[gen] Genus[mas, fem, neutr] Numerus[plural] Flexion[def]
die	Det Kasus[nom, akk] Genus[fem] Numerus[singular] Flexion[def]
kein	Det Kasus[nom] Genus[mas, neutr] Numerus[singular] Flexion[indef]
welchen	Det WPron[+] Kasus[dat] Genus[mas, fem, neutr] Numerus[plural] Flexion[def]
beliebige	Adj Kasus[nom] Genus[mas, fem, neutr] Numerus[singular] Flexion[def]
beliebige	Adj Kasus[nom, akk] Genus[mas, fem, neutr] Numerus[plural] Flexion[indef]
beliebiger	Adj Kasus[nom] Genus[mas] Numerus[singular] Flexion[indef]
computer	N Kasus[nom, dat, akk] Genus[mas] Numerus[singular]

Fortsetzung

computer	N Kasus[nom, gen, akk] Genus[mas] Numerus[plural]
regeln	N Kasus[nom, gen, dat, akk] Genus[fem] Numerus[plural]
antworten	N Kasus[nom, gen, dat, akk] Genus[fem] Numerus[plural]
ich	Pron Person[erste] Kasus[nom] Numerus[singular]
wer	Pron Wpron[+] Person[dritte] Kasus[nom]
wen	Pron Wpron[+] Person[dritte] Kasus[akk]
antworten	Verb Form[finit] Person[erste, dritte] Numerus[plural] Tempus[präs] Valenz[subj, datobj, präpobj auf]
hat	Verb Form[finit] Person[dritte] Numerus[singular] Valenz[partizip]
erzeuge	Verb Form[finit] Person[erste] Numerus[singular] Tempus[präs] Valenz[subj, akkobj]
erzeugt	Verb Form[finit] Person[dritte] Numerus[singular] Tempus[präs] Valenz[subj, akkobj]
erzeugt	Verb Form[partizip] Valenz[subj, akkobj]
geantwortet	Verb Form[partizip] Valenz[subj, datobj, präpobj auf]
auf	Präp Kasus[dat, akk]
nach	Präp Kasus[dat]

Abb. 32.14c: Das Lexikon zum ATN-Programm

EINGABE: nach welchen regeln hat der computer antworten erzeugt?

ABARBEITUNG:
(Knoten sind fettgedruckt, Kanten in Klammern, Register kursiv; die Rekursionstiefe ist durch Einrückung dargestellt.)

P:	Wort:	Konfigurationen:
1	nach	**Satz** (1) PUSH NP
		NP (13), (14), (15), (16) JUMP
		NP-Kern (17), (18) — Rücksetzen
		Satz (2) PUSH PP
		PP (20) CAT Präp
		Rektion[dat]
		Präposition[nach]
		LIFTR *Präposition*[nach]
2	welchen	**Präp-NP** (21) PUSH NP
		NP (13) CAT Det
		Kasus[dat]
		Genus[mas, fem, neutr]
		Numerus[plural]
		Flexion[def]
		Artikel[(Deter: welch)]
		LIFTR *Wpron*[+]
3	regeln	**NP-Kern** (17), (18) CAT N
		Kern[regel (Numerus: plural)]
		LIFTR *Kasus*[dat]
		LIFTR *Numerus*[plural]
		LIFTR *Person*[dritte]
		NP-Ende (19) POP
		Nominalphrase[regel (Numerus: plural) (Deter: welch)]
		LIFTR *Wpron*[+]
		PP-Ende (22) POP
		PräpPhrase[nach (regel (Numerus: plural) (Deter: welch))]
		Satz[W-frage]
4	hat	**FinVerb** (4) CAT Verb
		Verb[perfekt]
		Tempus[F]
		Valenz[partizip]
		Kongruenz[dritte, singular]

Fortsetzung

5	der	**Ergänzung** (5) PUSH NP
		NP (13) CAT Det
		Kasus[nom]
		Genus[mas]
		Numerus[singular]
		Flexion[def]
		Artikel[(Deter: der)]
6	computer	**NP-Kern** (17), (18) CAT N
		Kern[computer (Numerus: singular)]
		LIFTR *Kasus* [nom]
		LIFTR *Numerus* [singular]
		LIFTR *Person* [dritte]
		NP-Ende (19) POP
		Subjekt[(Subjekt: computer(Numerus: singular) (Deter: der))]
7	antworten	**Ergänzung** (5) PUSH NP
		NP (13), (14), (15), (16) JUMP
		NP-Kern (17), (18) CAT N
		Kern[antworten (Numerus: plural)]
		LIFTR *Kasus*[nom, gen, dat, akk]
		LIFTR *Numerus*[plural]
		LIFTR *Person*[dritte]
		NP-Ende (19) POP
		GenObjekt[GenObjekt: antworten (Numerus: plural))]
		DatObjekt[(DatObjekt: antworten (Numerus: plural))]
		AkkObjekt[(AkkObjekt: antworten (Numerus: plural))]
8	erzeugt	**Ergänzung** (5), (6), (7) CAT Verb
		Tempus[perfekt]
		Verb[erzeugen]
9	?	**Satzzeichen** (9), (10) WRD „?“
		Valenz (11) JUMP
		Komplemente[(Subjekt: computer (Numerus: singular) (Deter: der))]
		Komplemente[(Subjekt: computer (Numerus: singular) (Deter: der)) (AkkObjekt: antworten (Numerus: plural))]
		Komplemente[(Subjekt: computer (Numerus: singular) (Deter: der)) (AkkObjekt: antworten (Numerus: plural)) (Adverbiale: nach (regel (Numerus: plural) (Deter: welch))]
		Satzende (12) POP

(Illokution: W-frage (Prädikat: erzeugen (Tempus: perfekt) (Subjekt: computer (Numerus: singular) (Deter: der)) (AkkObjekt: antworten (Numerus: plural)) (Adverbiale: nach regel (Numerus: plural) (Deter: welch))

Abb. 32.14d: ATN-Abarbeitungsbeispiel

Prädikate sind Identität (=) und Enthaltensein (ε) und das Vorliegen eines nicht-leeren Durchschnitts zweier Mengen (∩). Die Bedingung kann auf Register und sonstige Bestandteile von Konfigurationen Bezug nehmen und mittels logischer Junktoren (wenn dann, und, oder, nicht) beliebig komplex strukturiert sein. Trifft die Bedingung zu, so wird T ('true') als Wert zurückgegeben, sonst F ('false'). Ein Test schlägt fehl, wenn die Auswertung der Bedingung den Wert F ergibt.

(C) Durchführung von Aktionen:
SETR-Aktion: Setze das Register mit dem angegebenen Namen in der Registerliste auf den Wert der angegebenen Form.
ADDR-Aktion: Füge dem Register mit dem angegebenen Namen (das als Liste implementiert ist) die angegebene Form als weiteres Element hinzu. Auf diese Weise kann eine Menge von Merkmalen akkumuliert werden.
SENDR-Aktion: Dies ist eine Prä-Aktion, die nur an PUSH-Kanten benutzt wird. Setze das Register mit dem angegebenen Namen

auf der nächst niedrigeren Rekursionsstufe, wie sie gerade von PUSH eingerichtet wird, auf den Wert der angegebenen Form.
LIFTR-Aktion: Dies ist die inverse Aktion zu SENDR. Setze das Register mit dem angegebenen Namen auf der nächst höheren Rekursionsstufe auf den Wert der angegebenen Form.
HOLD-Aktion: Setze die angegebene Form als ein Vorkommen der angegebenen Konstituente auf die HOLD-Liste. Die HOLD-Liste ist eine globale Liste, die auf allen Ebenen zugänglich ist. Sie dient dazu, jederzeit auf Kontextinformationen zurückgreifen zu können. Insbesondere wird diese Information von der VIR-Kante benutzt, die eine früher aufgefundene (beispielsweise links extraponierte) Konstituente so behandelt, als sei sie an der späteren Stelle aufgetreten.

(D) Die Auswertung von sog. Formen innerhalb einer Aktion:

Die Variable * steht in der Regel für das lexikalische Element an der aktuellen Eingabeposition. Eine Ausnahme bildet ihre Verwendung in PUSH- und POP-Kanten: In den Aktionen einer PUSH-Kante, die nach der erfolgreichen Rückkehr aus einem Unternetz durchgeführt werden, identifiziere * mit dem von POP gelieferten Ergebnis der Abarbeitung des Unternetzes.
Bei GETF liefere für das aktuelle lexikalische Element den Wert des angegebenen Parameters, z. B. ein nach dem Lexikon zugeordnetes Merkmal für Numerus, Kasus, Person usw.
Bei GETR liefere den Inhalt des angegebenen Registers als Wert. Ist das Register leer, so liefere den Wert F ('false').
BUILDQ ist eine Funktion, mit der die Strukturbeschreibung aufgebaut wird. Sie erwartet einen Rahmen, d. i. ein Strukturfragment mit Konstanten und Sonderzeichen, sowie eine Reihe von Registernamen. Eines der am meisten gebrauchten Sonderzeichen ist '+'. Substituiere für die im Rahmen enthaltenen Vorkommen dieses Zeichens die Inhalte der angegebenen Register in der Reihenfolge ihres Auftretens. (Angenommen das Register Subj enthält '(NP computer)', das Register Verb '(V beantworten)' und das Register Obj '(NP fragen)'. 'BUILDQ (S + (VP + +)) Subj Verb Obj' ergibt dann den Ausdruck '(S (NP computer) (VP (V beantworten) (NP fragen)))'. Das Sonderzeichen '*' steht für das aktuelle Lexem in der Eingabe. Es wird direkt innerhalb des Rahmens von BUILDQ benutzt.
Als Wert zweier durch ∩ verknüpfter Formen liefere den Durchschnitt aus beiden. Ist der Durchschnitt leer, so liefere den Wert F („False").

Anmerkung. Der vorgestellte ATN-Parser wird den Phänomenen des Deutschen besser gerecht als die übrigen besprochenen Verfahren. Es wird die Kongruenz zwischen Subjekt und Verb berücksichtigt; die möglichen Satzglieder werden von der Valenz des Verbs abhängig gemacht; es wird der variablen Wortstellung Rechnung getragen, indem jedes Satzglied im Vorfeld (vor dem finiten Verb) auftreten kann; es werden Aussagesätze, Fragesätze und Sätze mit Hilfsverb erkannt. Um Rücksetzen möglichst zu vermeiden, werden die vorgefundenen Satzglieder zunächst gesammelt (in den Kanten 1, 2, 5, 6) und dann mit der im Lexikon angegebenen Valenz des Verbs verglichen (Kante 11). Dabei werden z. T. bisherige Ergebnisse nachträglich uminterpretiert, z. B. das Verhältnis zwischen Verb, Hilfsverb und Tempus in Kante 7. Das Erkennen der nominalen Satzglieder erfolgt bottom-up, d. h. es wird (in den Kanten 1 und 5) nur generell eine NP gesucht, die dann, je nach dem für sie ermittelten Kasus, interpretiert wird. Als Strukturbeschreibung wird ein Dependenzbaum aufgebaut, dessen Elemente je aus einer syntagmatischen Rolle und einem Lexem bestehen.

Abwandlungen. Der obige Algorithmus entspricht einem Vorgehen top-down, von links nach rechts, Tiefe zuerst, mit Rücksetzen. Es sind aber auch andere Abarbeitungen des Netzwerkes möglich, z. B. parallele oder teils sequentielle, teils parallele Übergänge von einer Konfiguration zur anderen (Woods 1973). Auch eine Organisation von unten nach oben sowie von innen nach außen (sog. Inselparsing) ist vorgeschlagen worden (Bates 1980, 47). Es können verschiedene Knoten in einem Netz als Eingänge dienen; auf diese Weise erhält man generellere und kompaktere Netzwerke. Rücksetzen kann vermieden werden, indem Registerinhalte noch verändert werden können, wenn sich die Unrichtigkeit einer früheren Annahme später herausstellt. Häufig wird der Linguist wissen, unter welchen Umständen der Parser in eine bestimmte Sackgasse geraten und zum Rücksetzen gezwungen sein wird. Statt der 'blinden' Abarbeitung des Rücksetzspeichers empfiehlt sich dann eine explizite Angabe im Netz, auf welche Alternative (ggf. unter welcher Bedingung) unmittelbar zurückgesetzt

werden soll (vgl. Koch 1979, 21 ff.). R. Burton 1976 hat ein System geschrieben, das aus einem ATN ein kompiliertes Programm erzeugt. Ein Interpreter übersetzt das ganze Netz in eine einzige LISP-Funktion, wobei die Knoten Sprungmarken werden, Funktionen für das Vorrücken und das Backtracking bereitgestellt werden und die Tests und Aktionen als LISP-Funktionen eingefügt werden. Die Kompilation dieser Funktion ergibt eine „ATN-Maschine", die Sätze akzeptiert und Strukturen ausgibt (Bates 1980, 50ff.). Eine wesentliche formale Erweiterung sind sog. kaskadierte ATNs (CATN, siehe Woods 1980). Ein CATN ist eine Sequenz von ATNs, wobei jedes folgende ATN seine Eingabe von der Ausgabe des vorhergehenden nimmt. Eine solche Ausgabe wird durch eine eigene Aktion an den Kanten (TRANSMIT-Aktion) erzeugt. Kann das nächste Netzwerk eine Eingabe nicht verarbeiten, so wird der Pfad im vorangehenden Netzwerk, der die Ausgabe erzeugt hat, für den weiteren Verlauf blockiert. Möglich ist z. B. daß ein erstes Netzwerk eine syntaktische Analyse durchführt und ein zweites eine semantische.

Kritik. Der Vorteil erweiterter Übergangsnetzwerke ist, daß man praktisch alles machen kann. Das ist aber zugleich ein Nachteil des Formalismus. Große Grammatiken sind wegen der prozeduralen Notation schwer überschaubar (siehe z. B. das Netzwerk für das System LUNAR in Bates 1980, 34—36). Der Transfer von Merkmalen mittels SETR, SENDR und LIFTR und die Unifikation der Inhalte vieler Register sind umständlich. ATNs sind das ganze Gegenteil der Lexikalisierung, welche derzeit ein Trend moderner Grammatiken ist (vgl. 4.2.1.).

4. Lexikonbasierte Parser

Die oben beschriebenen Verfahren beruhen auf *Regelgrammatiken. Die folgenden verwenden *lexikalisierte Grammatiken. Die den Elementen im Lexikon zugeordneten Kategorien sind hier so strukturiert, daß aus ihnen selbst die syntagmatische Verbindungsfähigkeit des Elements hervorgeht. In der Lexikonphase werden jedem Element der Eingabe die entsprechenden Kategorien zugeordnet. Anschließend werden fortlaufend die Beschreibungen der Segmente, die miteinander kompatibel sind, ermittelt und zum Aufbau einer komplexen Strukturbeschreibung benutzt. Es bietet sich an, dabei eine *Teilergebnistabelle (chart) zu verwenden.

4.1. Am Konstituenzprinzip orientierte Verfahren

Auf der Grundlage der lexikalischen Beschreibung wird versucht, die Teil-Ganzes-Beziehungen der Konstituenten innerhalb der Eingabe zu rekonstruieren.

4.1.1. Parser für Kategorialgrammatiken

Prinzip. Die Grundidee der Kategorialgrammatik, die ursprünglich für Logikformalismen gedacht war, ist die, daß ein Funktor auf bestimmte Argumente angewendet wird, und sich daraus eine Entität des gleichen oder eines neuen Typs ergibt. Z. B. ergibt die Anwendung eines Prädikats auf ein oder mehrere Argumente eine Aussage. Die Argumente können u. U. selbst wieder in Funktoren und Argumente zerlegt werden. Ein Funktor wird in der Kategorialgrammatik durch eine Kategorie repräsentiert, die aus den Kategorien der Argumente sowie der Kategorie, die das Ergebnis der Anwendung des Funktors ist, zusammengesetzt ist. Der Funktor enthält also die Information über die ganze syntaktische Konstruktion und wird dadurch gleichzeitig selbst klassifiziert. Elemente, die keine Funktoren sind, bekommen einfache Kategorien. Dieses Prinzip ist nun verallgemeinert und auf Konstituentenstrukturgrammatiken angewendet worden. Hier ist eine der unmittelbaren Konstituenten der Funktor. Seine Kategorie enthält die Kategorien der übrigen unmittelbaren Konstituenten sowie die Kategorie der übergeordneten Konstituente. Da die lexikalischen Einheiten sämtliche syntaktische Information enthalten, ist das Analyseverfahren denkbar einfach. Es besteht darin, daß die zusammengesetzten Kategorien fortgesetzt um die passenden Kategorien der Elemente in der Umgebung gekürzt werden, bis nur noch eine Kategorie für die gesamte Eingabe übrig bleibt (wobei gefordert werden kann, daß sich diese mit der ausgezeichneten Kategorie der Grammatik bedeckt). Die Kürzungen entsprechen den Zusammenfassungen von unmittelbaren Konstituenten in einem regelgesteuerten bottom-up Parser (vgl. 1.2.). Die Abfolge von Funktor und Argumenten sind ebenfalls der Kategorie des Funktors zu entnehmen. Normalerweise wird ein Funktor

in einem Schritt auf genau ein Argument angewendet (die Zusammenfassungen sind also binär). Ein nach links oder rechts geneigter Schrägstrich zeigt in der Kategorie des Funktors an, daß das betreffende Argument unmittelbar links bzw. rechts vom Funktor stehen muß. Klammern umschließen zusammengesetzte Kategorien, die in bezug auf die Kürzungen eine Einheit darstellen sollen. Die allgemeine Regel für eine Reduktion mit einer rechts stehenden Konstituente (forward cancelling) lautet:

(R-1) $\alpha/\beta, \beta \Rightarrow \alpha$

Die für eine Reduktion mit einer links stehenden Konstituente (backward cancelling) lautet:

(R-2) $\beta, \beta\backslash\alpha \Rightarrow \alpha$

Dabei sind α und β einfache oder zusammengesetzte Kategorien. Zur Abarbeitung bietet sich ein dem Cocke-Algorithmus (vgl. 2.2.1.) entsprechendes Verfahren an. (Hays 1967, 142f.; Uszkoreit 1986)

Zusammenhang mit Ersetzungsgrammatiken. Es ist möglich, jede kontextfreie Phrasenstrukturgrammatik in eine schwach äquivalente Kategorialgrammatik zu übersetzen. Dazu muß die Grammatik zunächst so umgeformt werden, daß jede Regel ein terminales Element enthält, das als Funktor in der entsprechenden Konstituente dienen kann. Diese Bedingung ist z. B. in der *Greibach-Normalform, die der prädiktiven Analyse zugrunde liegt (vgl. 1.1.3.), per definitionem erfüllt. Daher gehen wir von ihr im folgenden Beispiel aus. Die einfache Übersetzungsvorschrift lautet in diesem Fall (vgl. Kratzer/Pause/von Stechow 1973, 240) wie folgt. Nach Maßgabe eine jeden Regel der Form

$X \rightarrow a\, Y_1\, Y_2 \ldots Y_n$

GRAMMATIK G8:

Übersetzung der terminalen Kategorien von G2 (Greibach-Normalform) in Kategorien einer Kategorialgrammatik:

n	=	S/VP, NP, N
det	=	S/VP/N, S/VP/AN, NP/N, NP/AN
vi	=	VP
vt	=	VP/NP, VP/PP/NP
präp	=	PP/NP
adj	=	AN/N

Lexikon:
wie G1, wobei aber die terminalen Kategorien entsprechend nebenstehender Übersetzungsvorschrift ersetzt worden sind.

EINGABE: computer verarbeiten eingaben nach regeln

TEILERGEBNISTABELLE (chart):

0	1	2	3	4	5
computer	verarbeiten	eingaben	nach	regeln	
S/VP NP N	VP/NP VP/PP/NP	S/VP NP N	PP/NP	S/VP NP N	
	VP VP/PP		PP		
	VP				
S					

Abb. 32.15: Parser für Kategorialgrammatiken

weise allen Elementen im Lexikon statt der Kategorie a die Kategorie

$X/Y_n/\ldots/Y_2/Y_1$

zu. Aufgrund einer Regel der Form

$X \rightarrow a$

weise allen Elementen statt der Kategorie a die Kategorie X zu.

Das Ergebnis ist eine Kategorialgrammatik, die nur Kürzungen eines Funktors mit einfachen rechten Konstituenten kennt. Man benötigt daher hier keine Klammern.

Voraussetzungen:

(V-1) Eine Kategorialgrammatik in Form eines Lexikons, in welchem jedem lexikalischen Element der Sprache eine Menge von einfachen oder zusammengesetzten Kategorien zugeordnet ist.

(V-2) Eine Teilergebnistabelle (chart), deren Einträge je aus eine Menge von Kategorien bestehen und denen je ein Segment der Eingabe mit einer bestimmten linken und rechten Grenze zugeordnet ist. (Vgl. die Analysematrix zum Cocke-Algorithmus in 2.2.1.)

Algorithmus:

(A-1) Lexikonphase: Erzeuge zu jedem Element der Eingabe einen Eintrag in der Teilergebnistabelle, bestehend aus der Menge der im Lexikon verzeichneten Kategorien. Ordne dem Eintrag als linke Grenze die Position des Elements minus 1 und als rechte Grenze die Position des Elements zu.

(A-2) Reduktion: Prüfe für alle bisherigen Einträge in der Teilergebnistabelle, ob sie eine zusammengesetzte Kategorie enthalten, die sich mit der Kategorie eines rechts angrenzenden Segments, d. i. eines weiteren Eintrags, dessen linke Grenze gleich der rechten des ersteren ist, nach (R-1) kürzen läßt. Wenn ja, so füge die reduzierte Kategorie einem Eintrag hinzu, dessen linke Grenze gleich der linken des ersten und dessen rechte Grenze gleich der rechten des zweiten Eintrags ist. Läßt sich kein weiterer Eintrag erzeugen, so weiter (A-3). (Bei zweiseitigen Kategorialgrammatiken ist ein analoger Test nach (R-2) durchzuführen.)

(A-3) Erfolgsprüfung: Befindet sich in der Teilergebnistabelle ein Eintrag mit der linken Grenze gleich 0 und der rechten Grenze gleich der Zahl der Eingabeelemente, so akzeptiere die Eingabe. Andernfalls weise sie zurück. (Darüber hinaus kann gefordert werden, daß die Kategorie in diesem Eintrag das ausgezeichnete Symbol der Grammatik ist.)

Herstellung der Strukturbeschreibung. Ein Phrasenstrukturbaum entsteht, wenn bei jeder Kürzung ein neuer Knoten gebildet wird, der mit der gekürzten Kategorie etikettiert wird, und ihm die Knoten mit dem Funktor und dem Argument untergeordnet werden. Bemerkenswert ist, daß der erzeugte Baum demjenigen entspricht, der von einer Grammatik in Chomsky-Normalform generiert wird (wie sie üblicherweise dem Cocke-Algorithmus zugrunde gelegt wird) und nicht der Struktur, die eine prädiktive Grammatik erzeugt, obwohl die Kategorialgrammatik aus letzterer abgeleitet wurde.

Anwandlungen. Verglichen mit der prädiktiven Analyse hat das obige Verfahren den Nachteil, daß es nicht linear vom Satzanfang aus fortschreitet. Man könnte sich auch eine Kürzungsregel folgender Form vorstellen, die eine Abarbeitung von links nach rechts erlauben würde:

(R-3) $\alpha/\beta, \beta/\mu \Rightarrow \alpha/\mu$

Soweit die Kategorien ausschließlich aus einer Greibach-Normalform-Grammatik abgeleitet worden sind, deckt sich die Abarbeitung nach (R-3) völlig mit der der prädiktiven Analyse. Für weniger eingeschränkte Kategorialgrammatiken ist (R-3) jedoch wahrscheinlich nicht brauchbar. In Hausser 1986 wird ein linksassoziativer Parser für Kategorialgrammatiken vorgestellt, der stets die Kategorie des nächsten Wortes mit der des bis dahin analysierten Satzanfanges kürzt. Die Art, wie dies geschieht, wird jedoch durch eine Menge von Regeln gesteuert, die auf die konkreten Kategorien Bezug nehmen. Es handelt sich also um eine Mischform zwischen lexikalisierter Grammatik und Regelgrammatik.

4.2. Parsing nach dem Slot-und-Filler-Prinzip

Eine weitere Möglichkeit, einen Parser zu organisieren, ist es, eine Menge von Rahmen (frames) vorzugeben, in die aktuelle Eingaben eingepaßt werden. Ein Rahmen enthält einerseits eine vorgegebene Struktur und an-

dererseits Leerstellen (slots) für noch ungeklärte Teile. Letztere werden geklärt, indem eine passende Besetzung (filler) für die Leerstelle gefunden wird. Ein klassischer Fall dieser Organisation von Wissenserwerb ist der Fragebogen. Von der *Mustererkennung, als dem parallelen Abgleichen zweier Symbolsequenzen unterscheidet sich der *Slot-Filler-Ansatz durch die Asymmetrie zwischen den Einheiten, die den Rahmen schaffen, und solchen, die den Rahmen ausfüllen. Diese Sicht deckt sich mit der alten Unterscheidung zwischen gesättigten und ungesättigten Ausdrücken, die in Form der Valenz und Rektion Eingang in die Grammatik gefunden hat. Das Slot-Filler-Prinzip ist am besten für das Parsen von Dependenzgrammatiken geeignet.

4.2.1. Chart-Parser für lexikalisierte Dependenzgrammatiken

Prinzip. Ziel ist der Aufbau eines Dependenzbaum als Analyseergebnis. Jeder Knoten in diesem Baum ist mit dem Lexem eines Elements der Eingabe etikettiert. Es steht nichts im Wege, jeden Knoten mit weiteren Etiketten zu versehen, z. B. mit funktional-syntaktischen und morpho-syntaktischen Kategorien. Das strukturiende Prinzip der *Dependenzgrammatik ist die Beziehung zwischen Lexemen und Ergänzungen, und nicht etwa nur zwischen Lexemen (bzw. Wörtern), wie häufig angenommen wird. Die Möglichkeit oder Notwendigkeit von Ergänzungen hängt von der lexikalischen Bedeutung der Wörter ab; ein Wort, das eine Relation bezeichnet, verlangt z. B. nach Bezeichnungen für die Entitäten, die in der besagten Relation stehen. Die syntagmatische Funktion der Ergänzungen läßt sich leicht konkret angeben. So gehört zum Begriff schlafen der Schläfer, zum Begriff ärgern der Verärgerte und das Ärgerliche. Diese Rollen werden im Satz von Satzgliedern ausgefüllt, die aus einem einzigen Wort wie auch aus umfangreichen Phrasen bestehen können. Z. B. können 'Fritz', 'der Mann auf der Bank', 'Wer die ganze Nacht durchgearbeitet hat' alle die Funktion übernehmen, im Zusammenhang mit dem Verb 'schläft' den Schläfer zu bezeichnen. In Dependenzbäumen sind für Phrasen dennoch keine Knoten vorgesehen, weil ihre interne Struktur wiederum eine Angelegenheit von Lexemen und ihren Ergänzungen ist, und somit Phrasen selbst wieder als Dependenzbäume repräsentiert werden können. Die Dependenzrelation besteht formal also zwischen regierenden Knoten und dependenten Teilbäumen, und nicht etwa nur zwischen einzelnen Knoten. (Unter einem Teilbaum verstehen wir im folgenden jeden Knoten und alle ihm unmittelbar oder mittelbar untergeordnete Knoten.) Abstrahiert man von den konkreten Rollen der Ergänzungen, so gelangt man zu grammatischen Funktionen wie Subjekt, Objekt usw. Diese werden als Etiketten einzelnen Knoten im Dependenzbaum hinzugefügt. Sie gelten aber eben nicht nur für diesen Knoten, sondern für den von ihm dominierten Teilbaum.

Für jede Relation zwischen einem Knoten und einem abhängigen Teilbaum in der angestrebten Dependenzdarstellung ist ein Rahmen vorzusehen, der aus dem Knoten und einer Leerstelle für den Teilbaum besteht, wobei die Leerstelle ein Muster für den dominierenden Knoten im Teilbaum darstellt. Nach dem Slot-Filler-Prinzip werden komplexere Bäume dadurch aufgebaut, daß die dominierenden Knoten von Teilbäumen in passende Leerstellen eingefügt werden, wobei die von ihnen selbst abhängigen Knoten erhalten bleiben. Anschließend wird der so entstandene Baum wieder in andere Leerstellen eingesetzt, und so fort, bis schließlich ein Dependenzbaum entstanden ist, der für jedes Element in der Eingabe genau einen Knoten enthält.

Unifikation. Sowohl der dominierende Knoten wie auch alle Leerstellen werden durch *komplexe Kategorien morpho-syntaktisch näher bestimmt. Aus den Werten der Parameter in einer Leerstelle und denen einer potentiellen Besetzung wird der Durchschnitt gebildet. Dieser darf für keinen Parameter leer sein. Parameter, die in Kongruenz stehen, werden in der Leerstelle markiert (im unten folgenden Beispiel durch 'C' als Wert). In diesem Fall müssen die aus dem Vergleich von Slot und Filler gewonnenen Werte mit den entsprechenden Parametern im dominierenden Knoten ebenfalls einen nicht-leeren Durchschnitt bilden. Kongruenz zwischen nebengeordneten Teilbäumen wird erzwungen, indem die Werte an den gemeinsamen dominierenden Knoten weitergeben werden und dort einen Durchschnitt bilden müssen. Auf die gleiche Weise ist es möglich, Charakteristika von Teilbäumen an ihren dominierenden Term weiterzureichen. Von diesen kann dann die Besetzbarkeit einer Leerstelle abhängig gemacht werden.

Lexikonorganisation. Jedem Eingabesegment müssen ein Lexem, morpho-syntaktische Kategorien und ggf. Leerstellen zugeordnet werden. Es empfiehlt sich, dies in zwei Schritten zu tun. Im ersten Schritt wird der dominierende Knoten eines Dependenzbaumes gebildet, der das Lexem und die der Wortform entsprechende Kategorie enthält. Dies geschieht nach Maßgabe eines morpho-syntaktischen Lexikons. Vielen Wortformen kommt dasselbe Lexem zu. Von diesem aber hängt in erster Linie die Ergänzungsfähigkeit ab. Es gibt daher einen zweiten Datenbestand, der Verweise von Lexemen auf Ergänzungstypen enthält. In einem dritten Teil des Lexikons sind für jeden Ergänzungstyp vorgefertigte Leerstellen mit entsprechenden funktional-syntaktischen Kategorien (Rollen) gespeichert. Alternative Leerstellen (mit derselben Rolle aber unterschiedlichen morpho-syntaktischen Anforderungen) werden als Disjunktion notiert. Im zweiten Schritt werden aufgrund der Verweise zum betreffenden Lexem die vorgefertigten Leerstellen herausgesucht und dem dominierenden Knoten untergeordnet. Der resultierende Dependenzbaum mit Leerstellen wird in der Teilergebnistabelle (chart) gespeichert.

Chart. Die Teilergebnistabelle enthält zunächst Dependenzbäume zu den kleinsten Eingabesegmenten, davon einige mit Leerstellen. Im Verlauf des Slot-Filling-Prozesses werden Bäume erzeugt, die zu immer größeren Segmenten in der Eingabe gehören. Auch diese werden in der Teilergebnistabelle verwaltet. Gegenüber den oben besprochenen Charts, deren 'Bögen' mit Kategorien für Konstituenten oder mit Konstitutentenstrukturbäumen etikettiert sind (vgl. den Earley-Parser in 2.1.1. und den Cocke-Algorithmus in 2.2.2.) besteht die Neuerung also darin, daß es jetzt Bäume mit Leerstellen sind, die jeweils ein Eingabesegment überspannen. Die Leerstellen aber bestimmen die Aktionen des Parsers; somit handelt es sich um eine Art von 'active chart parsing'. Eine zweite Neuerung ist die, daß in der Chart auch diskontinuierliche Konstituenten zu verwalten sind. Ein Vorteil des Slot-Filler-Ansatzes ist es nämlich, daß die Besetzung für eine Leerstelle nicht unmittelbar an das mit dem Rahmen etikettierte Segment anschließen muß. Damit ist das Bild des Bogens von der linken zur rechten Grenze eines Segment hinfällig. Um die von einem Teilergebnis abgedeckte Spanne in der Eingabe zu bezeichnen, verwenden wir Bitketten, wobei für jede Position in der Eingabe ein Bit vorgesehen ist. '1' steht für ein Element das Teil des betreffenden Segments ist, '0' für ein Element, das nicht Teil des Segments ist. '101' ist z. B. die Repräsentation für ein Segment, das aus dem ersten und dritten Element in der Eingabe geformt ist. Bevor eine potentielle Besetzung mit einer Leerstelle verglichen wird, wird der Durchschnitt zwischen den Bitketten beiden Bäume gebildet. Dieser Durchschnitt muß leer sein, denn andernfalls decken die Bäume teilweise dieselben Segmente ab, und damit kann einer nicht mehr Ergänzung zum anderen sein. Nachdem ein Filler in ein Slot eingefügt worden ist, wird die Union der Bitstrings beider Segmente gebildet und als Segmentmarkierung dem neuen Baum zugeordnet. Die Eingabe wird akzeptiert, wenn es wenigstens ein Segment gibt, dem eine Bitkette von der Länge der Eingabe entspricht, die nur noch aus '1' besteht. Der Parser durchläuft die Teilergebnistabelle in mehreren Zyklen. Zunächst werden nur kontinuierliche Segmente gebildet, d. h. solche die zwischen der ersten und letzten Eins keine Null enthalten. Anschließend folgt ein Durchlauf, in dem aus den bisherigen Zwischenergebnissen auch diskontinuierliche Segmente gebildet werden. Schließlich werden diese diskontinuierlichen Segmente wieder kontinuierlich zusammengefügt, und so fort.

Positionsmerkmale. Die verschiedenen Stellungsverhältnisse in der Eingabekette zählen im hier zugrundegelegten Dependenzformalismus zu den morpho-syntaktischen Merkmalen und werden durch explizite Parameter und Werte an den Knoten der Dependenzbäume notiert. Ein Stellungsmerkmal gilt immer für das einem abhängigen Teilbaum zugeordnete Segment relativ zu dem Segment des dominierenden Knotens. Es hat sich gezeigt, daß sich Stellungsregularitäten relativ zur Dependenzstruktur besser formulieren lassen, als in einem Phrasenstrukturmodell, und zwar deshalb, weil verschiedene Typen von Stellungsparametern eingeführt werden können, während die Phrasenstrukturgrammatik nur die eine Relation der Konkatenation kennt. Stellungsparameter bilden einen Teil der morpho-syntaktischen Kategorie, mit der jede Leerstelle versehen ist. Sie unterliegen ebenso der Unifikation wie die übrigen Merkmale. Nur stammen ihre Werte nicht aus dem Lexikon, sondern werden aus den aktuellen Verhältnissen in der Eingabe be-

rechnet. Dies geschieht auf der Grundlage der Bitketten der beteiligten Segmente in der Chart. Der unten benutzte Parameter 'Folge' hat z. B. den Wert 'links', wenn das letzte Bit des Fillers dem ersten Bit des Leerstellen eröffnenden Elements vorangeht; der Parameter 'Position' hat den Wert n, wenn die Bitfolge die n-te unter den Bitketten für alle dependenten Teilbäume zum selben dominierenden Element ist. Es soll hier darauf verzichtet werden, die verschiedenen Abfolgen, die sich auf der Grundlage zweier Bitketten definieren lassen, formal einzuführen. Sie werden vom Parser generell zur Verfügung gestellt, und der Linguist kann bei der Wahl seiner Kategorie für eine konkrete Grammatik auf sie zurückgreifen. (Literatur zum vorliegenden Verfahren: Hellwig 1978a und 1978b; Hellwig 1980; Hellwig 1986; Hellwig 1988; Literatur zu anderen Dependenzparsern: Starosta/Nomura 1986; Nelimarkka/Jäppinen/Lehtola 1985; Jäppinen/Lehtola/Valkonen 1986.)

Voraussetzungen:

(V-1) ein morpho-syntaktisches Lexikon, in dem den Wortformen der Sprache je ein Lexem und eine komplexe morpho-syntaktische Kategorie zugeordnet wird.

(V-2) eine Menge von Ergänzungsmustern mit vorgefertigten Lehrstellen für jeden Typ von Ergänzung.

(V-3) eine Menge von Verweisen von Lexemen auf Ergänzungsmuster.

(V-4) eine Teilergebnistabelle, deren Einträge aus einem Bitstring zur Darstellung des Eingabesegments und einem Dependenzbaum bestehen.

(V-5) eine Variable F für den eine Leerstelle suchenden Baum, eine Variable S für einen Baum mit Leerstellen sowie je eine Variable A und E für den Anfang und das Ende des Bereichs in der Teilergebnistabelle, in dem gesucht wird. A_d sei eine weitere Variable für den Anfang des Suchbereichs.

Algorithmus:

(A-1) Lexikonphase: Bilde zu jeder Wortform in der Eingabe einen Baum, der aus einem Knoten mit dem Lexem und der morpho-symtaktischen Kategorie besteht, wie sie das morpho-syntaktische Lexikon angibt. Ordne diesem Knoten die Leerstellen all jener Ergänzungsmuster unter, auf die von dem Lexem aus verwiesen wird. (Mittels der Variablen '?' in den Ergänzungsmustern und entsprechenden Konstanten in den Verweisen können die Leerstellen individuell eingeschränkt werden.) Speichere die Ergebnisse als je eigene Einträge in der Teilergebnistabelle ab und ordne ihnen einen Bitstring der Länge p zu, wobei p die Position des Eingabeelements ist. Setze das p-te Bit auf '1' und alle anderen Bits auf '0'. Setze die Variablen F, S und A sowie A_d auf 1 und die Variable E auf die Anzahl der Einträge in der Teilergebnistabelle.

(A-2) Durchlauf mit kontinuierlichen Leerstellenbesetzungen: Die Variable F laufe von A bis E. Für jeden Wert von F laufe die Variable S von 1 bis E. Insofern S ungleich F, insofern der Durchschnitt der Bitketten von F und S leer und insofern F rechts oder links angrenzend an S, unifiziere den obersten Term des Baumes F in der Tabelle mit jeder Leerstelle des Baumes S. Für jede Leerstelle, für die dies erfolgreich ist, insertiere den Baum F in Baum S (lösche dabei alle Leerstellen, die F selbst noch enthält, sowie alle alternativen Leerstellen in S) und speichere den neuen Baum an Ende der Teilergebnistabelle als neuen Eintrag ab. Ordne diesem Baum die Union der Bitketten aus F und S zu. Ist F gleich E, und wurden keine neuen Bäume erzeugt, so weiter (A-3). Sonst setze A auf E, setze E auf den letzten neuen Eintrag und weiter (A-2).

(A-3) Durchlauf mit diskontinuierlichen Leerstellenbesetzungen: Setze A auf A_d. Weiter wie (A-2), nur mit der Bedingung, daß F und S nicht angrenzen. Wenn kein neuer Baum erzeugt wurde, so weiter (A-4). Sonst setze A_d auf E, setze E auf den letzten neuen Eintrag und weiter (A-2).

(V-4) Erfolgsprüfung: Gebe alle Dependenzbäume als Ergebnis aus, denen ein Bitstring von der Länge der Zahl der Eingabeelemente mit nur '1' zugeordnet ist. Gibt es keinen solchen Eintrag in der Teilergebnistabelle, so weise die Eingabe zurück.

Anmerkung. Bis einschließlich (14) in Abb. 32.16b bildet der Parser kontinuierliche Zwischenergebnisse. (15) ist das Ergebnis ei-

GRAMMATIK G9 (äquivalent zu G7 in Abb. 32.14a)
(Syntagmatische Rollen sind durch Großbuchstaben, Lexeme durch Kursivdruck, Leerstellen durch Unterstrich an Stelle des Lexems markiert. Die Disjunktion alternativer Leerstellen ist durch Komma angezeigt. Die morpho-syntaktischen Kategorien bestehen aus Parametern und in eckige Klammern gesetzten Werten. Alternative Werte sind durch Kommata getrennt. „C" bedeutet, daß der betreffende Wert mit den Werten des gleichen Parameters im übergeordneten Term zu unifizieren ist; dabei muß der Durchschnitt im Falle des Parameters Position leer und in allen Fällen nicht-leer sein.)

Morpho-syntaktisches Lexikon: Wie in Abb. 32.14c ohne die Valenzmerkmale.
(Das Lexem zum Hilfverb haben ist *'perfekt;* der Eintrag zum Satzzeichen '.' ist ILLOCUTION *aussage;* die Einträge zum Satzzeichen '?' sind ILLOCUTION *W-frage* und ILLOCUTION *E-frage.*)

Verweise vom Lexem auf Ergänzungsmuster:

computer	+ individuativ
regel	+ individuativ
antwort	+ individuativ
antworten	+ subjekt, + datobjekt, + präpobj auf, + adverbiale
erzeugen	+ subjekt, + akkobjekt, + adverbiale
'perfekt	+ subjekt, + partizip
auf	+ präpphrase
nach	+ präpphrase

Ergänzungsmuster:

(+ individuativ N
  (DETER — Det Kasus[C] Genus[C] Numerus[C] Flexion[C] WPron[C] Folge[links])
  (ATTRIB — Adj Kasus[C] Genus[C] Numerus[C] Flexion[C]) Folge[links])

(+ subjekt Verb Form[finit]
  (,(SUBJEKT — N Kasus[nom] Numerus[C] Person[C] Folge[links] Position[1] VerbStlg[zweit, C])
  (SUBJEKT — Pron Kasus[nom] Numerus[C] Person[C] WPron[C] Folge[links] Position[1] VerbStlg[zweit, C])
  (SUBJEKT — N Kasus[nom] Numerus[C] Person [C] Folge[rechts] Position[2] VerbStlg[front, zweit, C])
  (SUBJEKT — Pron Kasus[nom] Numerus[C] Person[C] Folge[rechts] Position[2] VerbStlg[front, zweit, C])))

(+ datobjekt Verb Form[finit]
  (,(DATOBJ — N Kasus[dat] WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]
  (DATOBJ — N Kasus[dat] Folge[rechts] Position[3, 4, 5] Verb StlG[front, zweit, C])))

(+ datobjekt Verb Form[partizip]
  (,(DATOBJ — N Kasus[dat] WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]
  (DATOBJ — N Kasus[dat] Folge[links] Position[3, 4, 5] VerbStlg[front, zweit, C])))

(+ akkobjekt Verb Form[finit]
  (,(AKKOBJ — N Kasus[akk] WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]
  (AKKOBJ — N Kasus[akk] Folge[rechts] Position[3, 4, 5] VerbStlg[front, zweit, C])))

(+ akkobjekt Verb Form[partizip]
  (,(AKKOBJ — N Kasus[akk] WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]
  (AKKOBJ — N Kasus[akk] Folge[links] Position[3, 4, 5] VerbStlg[front, zweit,C])))

(+ adverbiale Verb Form[finit]
  (,(ADVERBIAL — Präp WPron[C] Folge[links] Position[1] VerbStlg[zweit, C])
  (ADVERBIAL — Präp Folge[rechts] Position[3, 4, 5] VerbStlg[front, zweit, C])

(+ adverbiale Verb Form[partizip]
  (,(ADVERBIAL — Präp WPron[C] Folge[links] Position[1] VerbStlg[zweit, C])
  (ADVERBIAL — Präp Folge[links] Position[3, 4, 5] VerbStlg[front, zweit, C])

(+ präpobjekt? Verb Form[finit]
  (,(PRÄPOBJ — ? Präp WPron[C] Folge[links] Position[1] VerbStlg[zweit, C])
  (PRÄPOBJ — ? Präp Folge[rechts] Position[6] VerbStlg[front, zweit, C])

(+ präpobjekt? Verb Form[partizip]
  (,(PRÄPOBJ — ? Präp WPron[C] Folge[links] Position[1] VerbStlg[zweit, C])
  (PRÄPOBJ — ? Präp Folge[links] Position[6] VerbStlg[front, zweit, C])

(+ partizip
  (PRÄDTEIL — Verb Form[partizip] WPron[C] Folge[rechts] Position[C] VerbStlg[C]

(+ präpphrase
  (— N Kasus[C] WPron[C] Folge[rechts]))

(ILLOCUTION *aussage*
  PRÄDIKAT — Verb Form[finit] Folge[links] VerbStlg[zweit]))

(ILLOCUTION *W-frage*
  (PRÄDIKAT — Verb Form[finit] WPron[+] Folge[links] VerbStlg[zweit]))

(ILLOCUTION *E-frage*
  (PRÄDIKAT — Verb Form[finit] Folge[links] VerbStlg[front]))

Abb. 32.16a: Lexikalisierte Dependenzgrammatik

EINGABE: nach welchen regeln hat der computer antworten erzeugt?

CHART:

Nr.:	Positionen:	Eingabesegment:	Erklärung:
(1)	1	'nach'	(Lexikon)
	(*nach* Präp Kasus[dat] (__ N Kasus[C] WPron[C] Folge[rechts]))		
(2)	01	'welchen'	(Lexikon)
	(*welch* Det Kasus[dat] Genus[mas, fem, neutr] Numerus[plural] WPron[+] Flexion[def]		
(3)	001	'regeln'	(Lexikon)
	(*regeln* N Kasus[nom, gen, dat, akk] Genus[fem] Person[dritte] Numerus[plural] (DETER __ Det Kasus[C] Genus[C] Numerus[C] Flexion[C] WPron[C] Folge[links]) (ATTRIB __ Adj Kasus[C] Genus[C] Numerus[C] Flexion[C] Folge[links]))		
(4)	0001	'hat'	(Lexikon)
	('*perfekt* Verb Form[finit] Person[dritte] Numerus[singular] (,(SUBJEKT __ N Kasus[nom] Numerus[C] Person [C] Folge[links] Position[1] VerbStlg[zweit, C]) (SUBJEKT __ Pron Kasus[nom] Numerus[C] Person[C] WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]) (SUBJEKT __ N Kasus[nom] Numerus[C] Person[C] Folge[rechts] Position[2] VerbStlg[front, zweit, C]) (SUBJEKT __ Pron Kasus[nom] Numerus[C] Person [C] Folge[rechts] Position[2] VerbStlg[front, zweit, C])) (PRÄDTEIL __ Verb Form[partizip] WPron[C] Folge[rechts] Position[C] VerbStlg[C]))		
(5)	00001	'der'	(Lexikon)
	(*der* Det Kasus[nom] Genus[mas] Numerus[singular] Flexion[def])		
(6)	000001	'computer'	(Lexikon)
	(*computer* N Kasus[nom, dat, akk] Genus[mas] Person[dritte] Numerus[singular] (DETER __ Det Kasus[C] Genus[C] Numerus[C] Flexion[C] WPron[C] Folge[links]) (ATTRIB __ Adj Kasus[C] Genus[C] Numerus[C] Flexion[C] Folge[links]))		
(7)	0000001	'antworten'	(Lexikon)
	(*antwort* N Kaus[nom, gen, dat, akk] Genus[fem] Person[dritte] Numerus[plural] (DETER __ Det Kasus[C] Genus[C] Numerus[C] Flexion[C] WPron[C] Folge[links]) (ATTRIB __ Adj Kasus[C] Genus[C] Numerus[C] Flexion[C] Folge[links]))		
(8)	00000001	'erzeugt'	(Lexikon)
	(*erzeugen* Verb Form[partizip] (,(AKKOBJ __ N Kasus[akk] WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]) (AKKOBJ __ N Kasus[akk] Folge[rechts] Position[3, 4, 5] VerbStlg[front, zweit, C])) (,(ADVERBIAL __ Präp WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]) (ADVERBIAL __ Präp Folge[links] Position[3, 4, 5] VerbStlg[front, zweit, C])))		
(9)	000000001	'?'	(Lexikon)
	(ILLOCUTION *W-frage* (PRÄDIKAT __ Verb Form[finit] WPron[+] Folge[links] VerbStlg[zweit]))		
(10)	011	'welchen regeln'	(2) in (3)
	(*regel* N Kasus[dat] Genus[fem] Person[dritte] Numerus[plural] WPron[+] Flexion-[def] (DETER *welch* Det Kasus[dat] Genus[fem] Numerus[plural] WPron[+] Flexion[def] Folge[links]))		
(11)	000011	'der computer'	(5) in (6)
	(*computer* N Kasus[nom] Genus[mas] Person[dritte] Numerus[singular] Flexion[def] (DETER *der* Det Kasus[nom] Genus[mas] Numerus[singular] Flexion[def] Folge[links]))		

Fortsetzung s. Seite 430

Nr.:	Positionen:	Eingabesegment:	Erklärung:
(12)	00000011	'antworten erzeugt'	(7) in (8)
	(*erzeugen* Verb Form[partizip] VerbStlg[front, zweit] (AKKOBJ *antwort* N Kasus[akk] Genus[fem] Numerus[plural] Folge[links] Position[5] VerbStlg[front, zweit]) (,(ADVERBIAL __ Präp WPron[C] Folge[links] Position[1] VerbStlg[zweit, C]) (ADVERBIAL __ Präp Folge[links] Position[3, 4] VerbStlg[front, zweit, C])))		
(13)	111	'nach welchen regeln'	(10) in (1)
	(*nach* Präp Kasus[dat] WPron[+] (*regel* N Kasus[dat] Genus[fem] Numerus[plural] WPron[+] Flexion[def] Folge[rechts] (DETER *welch* Det Kasus[dat] Genus[fem] Numerus[plura] WPron[+] Flexion[def] Folge[links])))		
(14)	000111	'hat der computer'	(11) in (4)
	('*perfekt* Verb Form[finit] Person[dritte] Numerus[singular] VerbStlg[front, zweit] (SUBJEKT *computer* N Kasus[nom] Genus[mas] Person[dritte] Numerus[singular] Flexion[def] Folge[rechts] Position[2] VerbStlg[front, zweit] (DETER *der* Det Kasus[nom] Genus[mas] Numerus[singular] Flexion[def] Folge[links])) (PRÄDTEIL __ Verb Form[partizip] WPron[C] Folge[rechts] VerbStlg[C]))		
(15)	1110011	'nach welchen regeln antworten erzeugt'	(13) in (12)
	(*erzeugen* Verb Form[partizip] WPron[+] VerbStlg[zweit] (AKKOBJ *antwort* N Kasus[akk] Genus[fem] Numerus[plural] Folge[links] Position[5] VerbStlg[front, zweit]) (ADVERBIAL *nach* Präp Kasus[dat] WPron[+] Folge[links] Position[1] VerbStlg[zweit] (*regel* N Kasus[dat] Genus[fem] Numerus[plural] WPron[+] Flexion[def] Folge[rechts] (DETER *welch* Det Kasus[dat] Genus[fem] Numerus[plural] WPron[+] Flexion[def] Folge[links]))))		
(16)	11111111	'nach welchen regeln hat der computer antworten erzeugt	(15) in (14)
	('*perfekt* Verb Form(finit] Person[dritte] Numerus[singular] WPron[+] VerbStlg[zweit] (SUBJEKT *computer* N Kasus[nom] Genus[mas] Person[dritte] Numerus[singular] Flexion[def] Folge[rechts] Position[2] VerbStlg[front, zweit] (DETER *der* Det Kasus[nom] Genus[mas] Numerus[singular] Flexion[def] Folge[links])) (PRÄDTEIL *erzeugen* Verb Form[partizip] WPron[+] Folge[rechts]VerbStlg[zweit] (AKKOBJ *antwort* N Kasus[akk] Genus[fem] Numerus[plural] Folge[links] Position[5] VerbStlg[front, zweit]) (ADVERBIAL *nach* Präp Kasus[dat] WPron[+] Folge[links] Position[1] VerbStlg[zweit] (*regel* N Kasus[dat] Genus[fem] Numerus[plural] WPron[+] Flexion[def] Folge[rechts] (DETER *welch* Det Kasus[dat] Genus[fem] Numerus[plural] WPron[+] Flexion[def] Folge[links]))))		
(17)	111111111	'nach welchen regeln hat der computer antworten erzeugt?'	(16) in (9)
	(ILLOCUTION *W-frage* (PRÄDIKAT '*perfekt* Verb Form[finit] Person[dritte] Numerus[singular] WPron[+] Folge[links] VerbStlg[zweit] (SUBJEKT *computer* N Kasus[nom] Genus[mas] Person[dritte] Numerus[singular] Flexion[def] Folge[rechts] Position[rechts] Position[2] VerbStlg[front, zweit] (DETER *der* Det Kasus[nom] Genus[mas] Numerus[singular] Flexion[def] Folge[links])) (PRÄDTEIL (*erzeugen* Verb Form[partizip] WPron[+] Folge[rechts] VerbStlg[zweit] (AKKOBJ *antwort* N Kasus[akk] Genus[fem] Numerus[plural] Folge[links] Position[5] VerbStlg[front, zweit]) (ADVERBIAL *nach* Präp Kasus[dat] WPron[+] Folge[links] Position[1] VerbStlg[zweit] (*regel* N Kasus[dat] Genus[fem] Numerus[plural] WPron[+] Flexion[def] Folge[rechts] (DETER *welch* Det Kasus[dat] Genus[fem] Numerus[plura] WPron[+] Flexion[def] Folge[links]))))))		

Abb. 32.16 b: Teilergebnistabelle des Parsers für Dependenzgrammatiken

nes Durchlaufs mit diskontinuierlichen Leerstellenbesetzungen. (16) und (17) sind wieder kontinuierlich. Die Chart zeigt im übrigen nur einen Ausschnitt; der Leser möge die nicht aufgeführten Lesungen im Lexikon sowie die nicht-erfolgreichen Zwischenergebnisse ergänzen, um zu sehen, wie sie blockiert werden. Während ATNs prototypisch sind für die prozedurale Denkweise, ist der hier vorgeführte Parser ein Musterbeispiel für deklaratives Programmieren. Dabei zeigen sich die Vorteile, die eine Unifikationsgrammatik bietet. Der Transport von Merkmalen über viele Stufen hinweg aufgrund der 'C'-Markierung in den Leerstellen entspricht den Aktionen LIFTR und SENDR in ATNs, ist aber viel flexibler und überschaubarer. Veränderungen der Grammatik greifen aufgrund des lexikalistischen Ansatzes weniger in das Bestehende ein als bei ATNs. Es gibt durch Hinzunahme weiterer Ergänzungsmuster kaum unerwünschte Seiteneffekte. Der Chart-Parser für lexikalisierte Dependenzgrammatiken ist somit ein gutes Beispiel für die modulare Organisation von Datenverarbeitungsaufgaben.

5. Literatur (in Auswahl)

A. V. Aho/R. Sethi/J. D. Ullman 1986 · A. V. Aho/J. D. Ullman 1972—73 · A. Barr/E. Feigenbaum 1981 · M. Bates [1978] 1980 · I. Bátori 1981 a · J. Bear/L. Karttunen 1979 · G. L. Berry-Rogghe/H. Wulz 1978 · B. Billmeier 1979 · D. G. Bobrow/E. Fraser 1969 · D. G. Bobrow/R. M. Kaplan/M. Kay 1977 · H. G. Bohnert/P. O. Backer 1979 · W. Brecht 1979 · K. Brockhaus 1971 · K. Brockhaus/A. von Stechow 1971 · B. Brodda 1983 · W. J. H. J. Bronnenberg/H. C. Bunt/S. P. J. Landsbergen et al. 1979 · H. C. Bunt/G. O. thoe Schwartzenberg 1982 · E. Charniak 1982 · E. Charniak 1983 · E. Charniak/D. McDermott 1985 · E. Charniak/C. K. Riesbeck/D. McDermott 1979 · E. Charniak/Y. Wilks 1976 · N. Chomsky 1957 · N. Chomsky 1959 · N. Chomsky 1963 · N. Chomsky 1965 · N. Chomsky 1970 · N. Chomsky 1981 · N. Chomsky 1982 · T. Christaller 1985 · T. Christaller/D. M. Metzing 1979 · T. Christaller/D. M. Metzing 1980 · T. Christaller/D. Metzing 1983 · K. W. Church 1983 · A. Colmerauer 1978 · A. Correira 1980 · G. W. Cottrell/S. L. Small 1984 · S. Crain/J. Fodor 1985 · S. Crain/M. Steedman 1985 · G. DeJong 1979 a · A. De Roeck 1983 · R. Dietrich/W. Klein 1974 · B. Dostert/F. B. Thompson 1976 · D. R. Dowty/L. Karttunen/A. M. Zwicky 1985 · M. G. Dyer 1981 · J. C. Earley 1970 · H. Eggers/R. Dietrich/W. Klein et al. 1969 · M. Eimermacher 1986 · A. Eisele 1985 · P. Eisenberg 1976 · R. Evans 1985 · J. D. Fodor/L. Frazier 1980 · J. M. Foster 1971 · W. Frey 1984 · W. Frey/U. Reyle 1983 · J. Friedman/D. Warren 1978 · H. Gaifman 1965 · J. M. Gawron/J. King/J. Lamping et al. 1982 · G. Gazdar 1981 · G. Gazdar/E. Klein/G. K. Pullum et al. 1985 · G. Gazdar/G. K. Pullum 1982 · M. Gehrke 1982 · E. von Glaserfeld 1970 · G. Görz/C. Beckstein 1986 · S. L. Graham/M. A. Harrison/W. L. Ruzzo 1980 · R. Granger/K. J. Eiselt/J. K. Holbrook 1984 · S. A. Greibach 1964 · T. Griffiths/S. R. Petrick 1965 · R. Grishman 1976 · M. Gross 1984 · M. Gross/A. Lentin 1971 · H. Gust 1983 · H. Gust 1984 · U. Hahn 1984 · W. v. Hahn/W. Hoeppner/A. Jameson et al. 1980 · M. A. K. Halliday 1967 · P. K. Halvorsen 1983 · L. R. Harris 1980 · M. A. Harrison 1978 · R. Hausser 1986 · P. J. Hayes/G. Mouradian 1981 · D. G. Hays 1966 a · D. G. Hays 1966 b · D. G. Hays 1967 · P. Hellwig 1978 a · P. Hellwig 1978 b · P. Hellwig 1980 · P. Hellwig 1984 · P. Hellwig 1986 · G. G. Hendrix 1977 · G. Hirst 1981 · J. R. Hobbs 1982 · T. Hoekstra/H. van der Hulst/M. Moortgat 1980 · J. E. Hopcroft/J. D. Ullman 1969 · G. Horn 1983 · R. A. Hudson 1980 · K. Hudson 1984 · E. T. Irons 1963 · R. Jackendoff 1977 · H. Jäppinen/A. Lehtola/K. Valkonen 1986 · R. Johnson 1983 b · R. M. Kaplan 1972 · R. M. Kaplan 1973 · R. M. Kaplan/J. Bresnan 1982 · L. Karttunen 1984 · L. Karttunen/M. Kay 1985 · L. Karttunen/A. Zwicky 1985 · M. Kay 1973 · M. Kay 1977 · M. Kay 1985 · J. Kilbury 1984 · J. Kimball 1975 · M. King 1983 · U. Klenk 1980 · U. Klenk 1983 · U. Klenk/J. Mau 1979 · D. E. Knuth 1968 · D. Koch 1979 · A. Kratzer/E. Pause/A. von Stechow 1973—74 · J. Krems 1984 · S. Kuno 1965 · S. Kuno [1967] 1976 · S. Kuno/A. G. Oettinger 1963 · J. Kunze 1975 · J. Kunze 1981 · S. C. Kwasny/N. K. Sondheimer 1981 · J. Landsbergen 1981 · S. P. J. Landsbergen/R. J. H. Scha 1979 · J. H. Laubsch 1979 · M. Lebowitz 1983 a · E. Leinfellner/I. Steinacker/H. Trost 1984 · L. Lesmo/P. Torasso 1983 · R. K. Lindsay 1971 · A. Lötscher 1979 · H. D. Luckhardt 1985 · M. P. Marcus 1980 · W. D. Marslen-Wilson 1975 · O. Mayer 1978 · M. C. McCord 1980 · S. Mehl 1986 · M. Minsky 1968 · E. Nelimarkka/H. Jäppinen/A. Lehtola 1985 · F. G. Pagan 1981 · R. C. Parkinson/K. M. Colby/W. S. Faught 1977 · F. C. N. Pereira 1981 · F. C. N. Pereira/D. H. D. Warren 1980 · F. C. N. Pereira/D. H. D. Warren 1983 · P. S. Peters/R. W. Ritchie 1969 · R. Petrick 1965 · S. Petrick 1973 · B. Phillips 1983 · M. Pinkal 1980 · G. K. Pullum 1984 · G. K. Pullum/G. Gazdar 1982 · C. Reddig 1984 · C. Rieger 1976 · C. K. Riesbeck 1975 · C. K. Riesbeck 1982 · G. D. Ritchie 1983 · C. R. Rollinger 1984 · R. Root 1982 · L. Rostek 1979 · G. Sabah/M. Rady 1983 · P. J. Sallis 1978 · G. R. Sampson 1983 · R. C. Schank 1975 · R. C. Schank/M. Lebowitz/L. Birnbaum 1980 · R. C. Schank/R. P. Abelson 1977 · H. Schnelle 1984 · S. M. Shieber 1983 · S. M. Shieber 1984 · S. M. Shieber 1986 · C. L. Sidner 1985 · G. Silva/D. Dwiggins 1980 · R. F. Simmons 1984 · J Slocum 1981 · S. Small 1983 · S.

Small/C. Rieger 1982 · K. Sparck Jones/Y. Wilks 1983 · S. Starosta/H. Nomura 1986 · H. Stoyan/H. Wedekind 1983 · L. Tesnière 1959 · M. Thiel 1986 · H. Thompson/G. Ritchie 1984 · M. Tomita 1985 · M. Tomita 1987 · H. Uszkoreit 1986 · G. B. Varile 1983 · D. L. Waltz 1978 · D. S. Warren/J. Friedman 1982 · J. Weizenbaum 1966 · J. Weizenbaum 1967 · R. S. Wells 1947 · A. van Wijngaarden 1969 · A. van Wijngaarden 1974 · R. Wilensky 1981 · Y. Wilks 1975 · Y. Wilks 1976 · Y. Wilks 1978 · Y. Wilks 1983a · T. Winograd 1972 · T. Winograd 1976 · T. Winograd 1983 · W. A. Woods [1970] 1976 · W. A. Woods 1973 · W. A. Woods 1980 · A. Zwicky/J. Friedman/B. Hall et al. 1965

Peter Hellwig, Heidelberg
(Bundesrepublik Deutschland)