Peter Hellwig: Kurs Maschinelle Syntaxanalyse (Parsing)

Aufgaben und Lösungen 2

Aufgabe 1: Die Idee der generativen Grammatik

Gegeben seien die folgenden Sätzen der Sprache Jepeen mit ihrer englischen Bedeutung:

	Jepeen:	**Englisch:**
(1)	eu ee xener xoieu	I ate.
(2)	eu ee xenere ne ixe xoieu	I ate fish.
(3)	xo ne lezav xoixo	You will jump.
(4)	xo ne lezave ne solon xoixo	You will jump over a stone.
(5)	I ee lezave ne solon xoxie	He jumped over a stone.
(6)	xo xuite ne toe xoixo	You are hitting the chicken.
(7)	I xuite ne wetipu xoxie.	He hits your wife.
(8)	I xuit xoxie	He hits.
(9)	I xuite ne tetipu xoxie	He hits your brother.
(10)	I xuite ne toe loe ne wetifu	My husband is hitting the black chicken.
(11)	I ne xenere ne ixe ne tetifu	My brother will eat fish.
(12)	I ne xeneri ne ixe	The fish will be eaten.
(13)	I lezavi ne solon lavu.	The big stone is jumped over.

Aufgabe:

1. Ermitteln Sie die Morpheme (lexikalische sowie grammatische) in den Beispielsätzen.
2. Schreiben Sie ein Lexikon, in dem den Wörtern Kategorien zugeordnet werden. Es handelt sich um terminale Kategorien im Sinne der Phrasenstrukturgrammatik. Sie können die Symbole für die Kategorien selbst wählen.
3. Schreiben Sie die Produktionsregeln einer Phrasenstrukturgrammatik, welche zusammen mit dem Lexikon die Sätze (1) - (13) in Jepeen generiert.
4. Generieren Sie mit Ihrer Grammatik denjenigen Satz in Jepeen, der dem englischen Satz entspricht:

   Your husband ate the big chicken

   und dokumentieren Sie die Generierung in Form eines Phrasenstrukturbaums zum generierten Satz.

Aufgabe 2: Top-down Parser mit Rücksetzen (PT-1)

Gegeben sei das folgende Fragment einer Grammatik des Schwäbischen:

Regeln:		**Lexikon:**	***Hochdeutsch:***
(R-1)	S -> pron1 + aux1 + VP	aux1 = *han*	*habe*
(R-2)	VP -> NP2 + VPP	aux3 = *hot*	*hat*
(R-3)	VP -> VP + RelSatz	auxp = *kett*	*gehabt*
(R-4)	VPP -> verb	det = *a*	*ein*
(R-5)	VPP -> verb + auxp	n = *Kent*	*Kind*
(R-6)	Relsatz -> relpron + aux3 + VP	pron1 = *i*	*ich*
(R-7)	NP2 -> prono	prono = *oine*	*eine (Frau)*
(R-8)	NP2 -> det + n	relpron = *die*	*die*
		verb = {*kennt, kett*}	*gekannt, gehabt*

1. Konstruieren Sie nach der Beschreibung des Parsers PT-1 im Skript INPUT TABLE, WORKING SPACE und BACKTRACKING STORE für den Satz

"I han oine kennt kett die hot a Kent kett"

2. Beenden Sie die Aufgabe, wenn der Parser ein Ergebnis gefunden hat. Würde der Parser auch von alleine aufhören? Würde gewährleistet bleiben, dass der Parser wenigstens ein Ergebnis findet, wenn man die Reihenfolge der Regeln beliebig änderte?

Aufgabe 3: Top-down Parser (PT-1) rekursiv programmiert

1. Entnehmen Sie die Grammatik der Sprache Jepeen der Lösung der Aufgabe 1.

2. Folgen Sie dem Struktogramm für PT1 mit "recursive procedure calls" im Skript. Nachvollziehen Sie den Algorithmus, indem Sie einen Trace (vgl. Skript) für folgenden Satz erstellen

   "xo xener xoixo"

3. Wem das mit der Hand zu mühsam ist, darf das Programm auch implementieren, so dass es den Trace ausdruckt.

4. Bricht der Algorithmus nach einem ersten Ergebnis ab, oder ermittelt er alle möglichen Ergebnisse?

Aufgabe 4: Top-Down Parser mit paralleler Abarbeitung (PT-2)

1. Gegeben sei noch einmal die Grammatik der Sprache Jepeen aus der Lösung von Aufgabe 1
2. Folgen Sie der Beschreibung von PT-2 im Skript und zeigen Sie, wie der folgende Satz im WORKING SPACE abgearbeitet wird:

"I ee xenere ne lavu ne wetipu"

Aufgabe 5: Top-down Parser mit Greibach-Nomalform (PT-3)

Gegeben sei wieder die Grammatik des Schwäbischen:

Regeln:		**Lexikon:**	***Hochdeutsch:***
(R-1)	S -> pron1 + aux1 + VP	aux1 = *han*	*habe*
(R-2)	VP -> NP2 + VPP	aux3 = *hot*	*hat*
(R-3)	VP -> VP + RelSatz	auxp = *kett*	*gehabt*
(R-4)	VPP -> verb	det = *a*	*ein*
(R-5)	VPP -> verb + auxp	n = *Kent*	*Kind*
(R-6)	Relsatz -> relpron + aux3 + VP	pron1 = *i*	*ich*
(R-7)	NP2 -> prono	prono = *oine*	*eine (Frau)*
(R-8)	NP2 -> det + n	relpron = *die*	*die*
		verb = {*kennt, kett*}	*gekannt, gehabt*

1. Formen Sie diese Grammatik in eine Grammatik in Greibach-Normalform um, wie im Skript beschrieben.
2. Denken Sie sich einen Algorithmus und einen WORKING SPACE für eine parallele Abarbeitung der *predictions* aus ähnlich wie PT-2. Zeigen Sie, wie folgender Satz damit verarbeitet wird:

"I han oine kennt kett die hot oine kennt kett die hot a Kent kett"

Aufgabe 6: Top-Down Chart Parser nach Earley (PT-4)

Gegeben sei folgende Grammatik

Regeln	
(R-1)	**S -> NP VV**
(R-2)	**S -> NP VA**
(R-3)	**VV -> vi**
(R-4)	**VV -> vt NP**
(R-5)	**VA -> aux Vinf**
(R-6)	**Vinf -> Zuvi**
(R-7)	**Vinf -> NP Zuvt**
(R-8)	**Zuvi -> zu vi**
(R-9)	**Zuvt -> zu vt**
(R-10)	**NP -> pron**
(R-11)	**NP -> det n**
(R-12)	**NP -> adv det n**

Lexikon		
vi	=	*{singen, labern, schweigen}*
vt	=	*{haben, trinken, bezahlen}*
aux	=	*{haben, scheinen}*
zu	=	*{zu}*
det	=	*{die, meine, keine, drei}*
adv	=	*{noch, schon, erst}*
n	=	*{Schnäpse, Bouletten, Kekse}*
pron	=	*{wir, sie, manche, alle}*

1. Folgen Sie der Beschreibung von PT-4 im Skript und zeigen Sie, wie der folgende Satz durch den Predictor, Scanner und Completor mit Hilfe von divided productions in einer Chart abgearbeitet wird:

   "Wir haben noch drei Bouletten"

2. Es sei anschließend der folgende Satz zu parsen:

   "Wir haben noch drei Bouletten zu bezahlen"

   Ändern Sie die Working Table nur von der Stelle an, ab der die Sätze divergieren.

3. Welche Zeilen der Gesamtanalyse für den zweiten Satz, welche PT-4 nur einmal erzeugt, hätte ein Parser mit Backtracking (PT-1) zweimal erzeugt?

4. Bei der Analyse von *"Wir haben noch drei Bouletten"* kommt es bei dem vorgeschlagenen Algorithmus zu einem Durcheinander unter den Sektionen (d.h. die rechte Grenze der nächsten Zeilen steigt nicht immer gleichmäßig an.) Wann tritt solch ein Durcheinander auf und wie ist es zu erklären.?

Aufgabe 7: Bottom-Up Chart Parser nach Cocke (PT-5)

1. Gehen Sie von folgender Grammatik des Schwäbischen in Chomsky-Normalform aus. Fettgedruckte Kategorien markieren jeweils die dominierende Konstituente (den *Head*) in der Regel.

Regeln:

(R-1)	S -> pron1 + **V1**
(R-2)	V1 -> **aux1** + Vo
(R-3)	V1 -> **aux1** + Vp
(R-4)	V1-> **aux1** + Vr
(R-5)	Vo -> prono + **verb**
(R-6)	Vo -> No + **verb**
(R-7)	No -> det + **n**
(R-8)	Vp -> Vo + **auxp**
(R-9)	Vr -> **Vp** + RelSatz
(R-10)	Vr -> **Vo** + Relsatz
(R-11)	Relsatz -> relpron + **V3**
(R-12)	V3 -> **aux3** + Vo
(R-13)	V3 -> **aux3** + Vp
(R-14)	V3-> **aux3** + Vr

Lexikon:

aux1 = *han*
aux3 = *hot*
auxp = *kett*
det = *a*
n = *Kent*
pron1 = *i*
prono = *oine*
relpron = *die*
verb = {*kennt, kett*}

Hochdeutsch:

habe
hat
gehabt
ein
Kind
ich
eine (Frau)
die
gekannt, gehabt

(Die Subklassifikation von V und aux bedeutet: 1=erste Person, 3=dritte Person, p=plus-quam-perfekt, o=mit Objekt, r=mit Relativsatz - solche Merkmale könnte man natürlich besser mit komplexen Kategorien darstellen.)

2. Erstellen Sie Input Table und Working Table von PT-5, wie im Skript beschrieben, für den Satz

"I han oine kennt kett die hot a Kent kett"

3. Konstruieren Sie nach Anleitung des Skripts aus den Einträgen in der Working Table einen Dependenzbaum

Aufgabe 8: Parsing mit einer Kategorialgrammatik (ähnlich PT-5)

1. Gehen Sie von folgender (abgewandelter) Grammatik des Schwäbischen in Chomsky-Normalform aus. Fettgedruckte Kategorien markieren jeweils die dominierende Konstituente (den *Head*) in der Regel.

Regeln:		**Lexikon:**	***Hochdeutsch:***
(R-1)	S -> pron1 + **V1**	pron1 = *i*	*ich*
(R-2)	S -> pron3 + **V3**	pron3 = die	*die*
(R-3)	V1 -> **aux1** + Vo	aux1 = *han*	*habe*
(R-4)	V1 -> **aux1** + Vp	aux3 = *hot*	*hat*
(R-5)	V3 -> **aux3** + Vo	auxp = *kett*	*gehabt*
(R-6)	V3 -> **aux3** + Vp	prono = *oine*	*eine (Frau*
(R-7)	Vo -> prono + **verb**	det = *a*	*ein*
(R-8)	Vo -> No + **verb**	n = *Kent*	*Kind*
(R-9)	No -> det + **n**	verb = {*kennt, kett*}	*gekannt, gehabt*
(R-10)	Vp -> Vo + **auxp**		

2. Wandelt Sie die Regeln in kategorialgrammatische Kategorien um, die Sie den Lexikonelementen zuordnen.
3. Zeigen Sie wie folgende beiden Sätze (separat) geparst werden:

   i han oine kennt kett

   die hot a Kent kett

Aufgabe 9: Tabellengesteuerter Shift-Reduce Parser (PT-6)

Gegeben die folgende Grammatik des Schwäbischen:

Regeln:		Lexikon:	Hochdeutsch:
(R-1)	S -> pron1 aux1 VP	aux1 = *han*	habe
(R-2)	VP -> NP2 VPP	aux3 = *hot*	hat
(R-3)	VP -> VP RelSatz	auxp = *kett*	gehabt
(R-4)	VPP -> verb	det = *a*	ein
(R-5)	VPP -> verb auxp	n = *Kent*	Kind
(R-6)	Relsatz -> relpron aux3 VP	pron1 = *i*	ich
(R-7)	NP2 -> prono	prono = *oine*	eine (Frau)
(R-8)	NP2 -> det n	relpron = *die*	die
		verb = {*kennt, kett*}	gekannt, gehabt

1. Überführen Sie nach Anleitung des Skripts diese Grammatik in eine Control Table für den tabellen- gesteuerten Shift-Reduce Parser (bestehend aus Action Table und Goto Table)

2. Prüfen Sie Ihre Tabellen, indem Sie den folgenden Satz verarbeiten:

" I han oine kennt die hot a Kent kett"

Aufgabe 10: Deterministischer FTN-Parser für reguläre Ausdrücke (PT-7)

Ein Beispiel für reguläre Sprachen sind Grammatikformalismen. (Wohlgemerkt, die Syntax des Grammatikformalismus selbst ist regulär, nicht unbedingt die der Sprachen, welche die Grammatik beschreibt.) Grammatikformalismen sind normalerweise regulär, weil ihre Verarbeitung einfach sein soll. Ein interpretierender Parser muss ja nicht nur die natürliche Sprache parsen sondern auch die Grammatik, die er beim Parsen konsultiert.

Ein Beispiel für einen Grammatikformalismus ist z.B. G1 aus dem Skript::

Rules	
(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 -> prep NP

Lexicon		
vi	=	*{sleep, fish}*
vt	=	*{study, visit, see, enjoy}*
det	=	*{the, no, my, many}*
adj	=	*{foreign, beautiful}*
n	=	*{tourists, pyramids, friends, fish, cans, Egypt, we, they}*
prep	=	*{in, by, with}*

1. Beschreiben Sie zunächst die Konventionen für die Regeln und Lexikoneinträge von derartigen Phrasenstrukturgrammtiken mit eigenen Worten.

2. Halten Sie diese Konventionen in Form von je einem regulären Ausdruck für die Regeln und einem regulären Ausdruck für die Lexikoneinträge fest.

3. **Achtung**: Das Vokabular soll aus den Buchstaben (A ...Z a...z, 1 ... 0) und Sonderzeichen (->, =, {. }) bestehen, die im Formalismus prinzipiell verwendet werden können, nicht etwa aus den Kategorien und Wörtern einer konkreten Grammatik wie G1. Sie können Teilmengen des Vokabulars bilden und dafür eine Kategorie einführen, die Sie im regulären Ausdruck verwenden (vgl. (iii) in der Definition regulärer Ausdrücke im Skript.)

4. Konstruieren Sie zu dem regulären Ausdruck für die Regeln ein nicht-deterministisches endliches Übergangsnetzwerk. Überflüssige Epsilon-Kanten können Sie einsparen, aber Vorsicht - manche sind nötig, um nicht in falsche Schleifen zu geraten.

5. Erzeugen Sie aus dem nicht-deterministischen endlichen Übergangsnetzwerk eine Tabelle für einen deterministischen endlichen Erkenner.

6. Zeigen Sie an einer der Regeln der obigen Grammatik G1, dass der Erkenner korrekt funktioniert.

Aufgabe 11: Augmented Transition Network (ATN) Parser (PT-8)

Römische Zahlen sehen bekanntlich so aus:

I = 1	XIII = 13	XXIX = 29	CC = 200
II = 2	XIV = 14	XXX = 30	CD = 400
III = 3	XV = 15	XL = 40	D = 500
IV = 4	XVI = 16	L = 50	DC = 600
V = 5	XVII = 17	LIX = 59	M = 1000
VI = 6	XVIII = 18	LX = 60	1558 = MDLVIII
VII = 7	XIX = 19	LXX = 70	1625 = MDCXXV
VIII = 8	XX = 20	LXXX = 80	1804 = MDCCCIV
IX = 9	XXIII = 23	XC = 90	1900 = MCM
X = 10	XXIV = 24	C = 100	1962 = MCMLXII
XI = 11	XXVII = 27	CI = 101	1999 = MIM
XII = 12	XXVIII = 28	CL = 150	2000 = MM

Aufgabe:

1. Schreiben Sie nach dem Vorbild von PT-8 im Skript ein ATN-Programm, das römische Zahlen parst und als Output arabische Zahlen erzeugt.

2. Zeigen Sie anhand eines Traces (so wie im Skript) für die Zahl **MCMLXII**, dass Ihr Programm korrekt funktioniert.

Aufgabe 12: Slot-Filler Parser für Dependenzgrammatiken (PT-9)

Ein Parser nach dem Vorbild von PT-9 im Skript soll alle möglichen Kombinationen von Hilfsverben und Vollverb im Deutschen akzeptieren können und jeweils einen Dependenzbaum in Klammernotation ausgeben. An Hilfverben sollen vorkommen: *haben* und *sein* als Perfekt, *werden* als Futur, *werden* als Passiv, *können, dürfen, müssen* als Modalverben. Als Vollverb nehmen wir das Wort *analysieren*. Folgende Kombinationen sind möglich:

	Vollverb	**Passiv**	**Modal**	**Perfekt**	**Futur**
Futur	er wird analysieren	es wird analysiert werden	er wird analysieren können	er wird analysiert haben	
Perfekt	er hat analysiert	es ist analysiert worden	er hat analysieren können	er hat analysiert gehabt	
Modalverb	er kann analysieren	es kann analysiert werden			
Passiv	es wird analysiert				
Vollverb					

Es steht folgendes morphosyntaktische Lexikon zur Verfügung:

analysieren	lexem[analysieren] kategorie[verb] verbtyp[vollverb] form[infinitiv]
analysiert	lexem[analysieren] kategorie[verb] verbtyp[vollverb] form[partizip] perfekt[haben]
gekommen	lexem[kommen] kategorie[verb] verbtyp[vollverb] form[partizip] perfekt[sein]
wird	lexem[passiv'] kategorie[verb] verbtyp[passiv] form[finit]
werden	lexem[passiv'] kategorie[verb] verbtyp[passiv] form[infinitiv]
worden	lexem[passiv'] kategorie[verb] verbtyp[passiv] form[partizip] perfekt[sein]
kann	lexem[können] kategorie[verb] verbtyp[modal] form[finit]
können	lexem[können] kategorie[verb] verbtyp[modal] form[infinitiv] perfekt[haben]
darf	lexem[dürfen] kategorie[verb] verbtyp[modal] form[finit]
dürfen	lexem[dürfen] kategorie[verb] verbtyp[modal] form[infinitiv] perfekt[haben]
muß	lexem[müssen] kategorie[verb] verbtyp[modal] form[finit]
müssen	lexem[müssen] kategorie[verb] verbtyp[modal] form[infinitiv] perfekt[haben]
hat	lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[finit] perfekt[haben]

haben	lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[infinitiv] perfekt[haben]
gehabt	lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[partizip] perfekt[haben]
ist	lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[finit] perfekt[sein]
sein	lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[infinitiv] perfekt[sein]
gewesen	lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[partizip] perfekt[sein]
wird	lexem[futur'] kategorie[verb] verbtyp[futur] form[finit]

1. Schreiben Sie die benötigten Templates und weisen Sie Lexemen die Templates zu (d.h. machen Sie die passenden Valenzangaben).
2. Zeigen Sie für eine etwas komplizierteres Beispiel von Hilfsverbhäufung, wie der Slot-und-Filler Parser einen Dependenzbaum aufbaut.

Aufgabe 13: Evaluation, Komplexität

Vergleichen Sie Ihre Hausaufgaben. Versuchen Sie jeweils den Aufwand des betreffenden Parsers abzuschätzen (Zahl der Rechenschritte in Abhängigkeit von der Zahl der Eingabewörter, der Zahl der Regeln, der Zahl von Alternativen unter den Regeln u.a.)
Versuchen Sie die Prototypen auf einer Effizienzskala anzuordnen.

Lösung 1: Die Idee der generativen Grammatik

Gegeben folgende Sätze der Sprache Jepeen:

	Jepeen:	**Englisch:**
(1)	eu ee xener xoieu	I ate.
(2)	eu ee xenere ne ixe xoieu	I ate fish.
(3)	xo ne lezav xoixo	You will jump.
(4)	xo ne lezave ne solon xoixo	You will jump over a stone.
(5)	I ee lezave ne solon xoxie	He jumped over a stone.
(6)	xo xuite ne toe xoixo	You are hitting the chicken.
(7)	I xuite ne wetipu xoxie.	He hits your wife.
(8)	I xuit xoxie	He hits.
(9)	I xuite ne tetipu xoxie	He hits your brother.
(10)	I xuite ne toe loe ne wetifu	My husband is hitting the black chicken.
(11)	I ne xenere ne ixe ne tetifu	My brother will eat fish.
(12)	I ne xeneri ne ixe	The fish will be eaten.
(13)	I lezavi ne solon lavu.	The big stone is jumped over.

Die Morpheme (lexikalische sowie grammatische) in den Beispielsätzen:

Jepeen	***Englisch***	***vorgekommen in Satz***
SUBSTANTIVE		
ixe	fish	2,11,12
solon	stone	4,5,13
toe	chicken	6,10
teti-fu	my brother	11
teti-pu	your brother	9
weti-fu	my husband	10
weti-pu	your wife	7
Possessiv-Endungen		
-fu	my	
-pu	your	
ARTIKELWÖRTER		
ne	-, the, a	2,4,5,6,7,9,10,11,12,13
PRONOMINA (immer am Satzanfang, sog. *Clitics*)		
eu	I	1,2
xo	you	3,4,6
i	he, she, it	5,7-13

PRONOMINA (am Satzende)

xoi-eu	I	1,2
xoi-xo	you	3,4,6
xox-ie	he	5,7,8,9

Person in den Clitics und Pronomina

eu, -eu	I
xo, -xo	you
i, -ie	he, she, it

VERBEN

lezav, lezav-e, lezav-i	jump	3,4,5,13
xener, xener-e, xener-i	eat	1,2,11,12
xuit, xuit-e	hit	6,7,8,9,10

Genus des Verbs

-e	mit Object
-i	Passiv
-	sonst

HILFSVERBEN

ee	past	1,2,5
ne	future	3,4,11,12
-	present	6-10,13

ADJEKTIVE

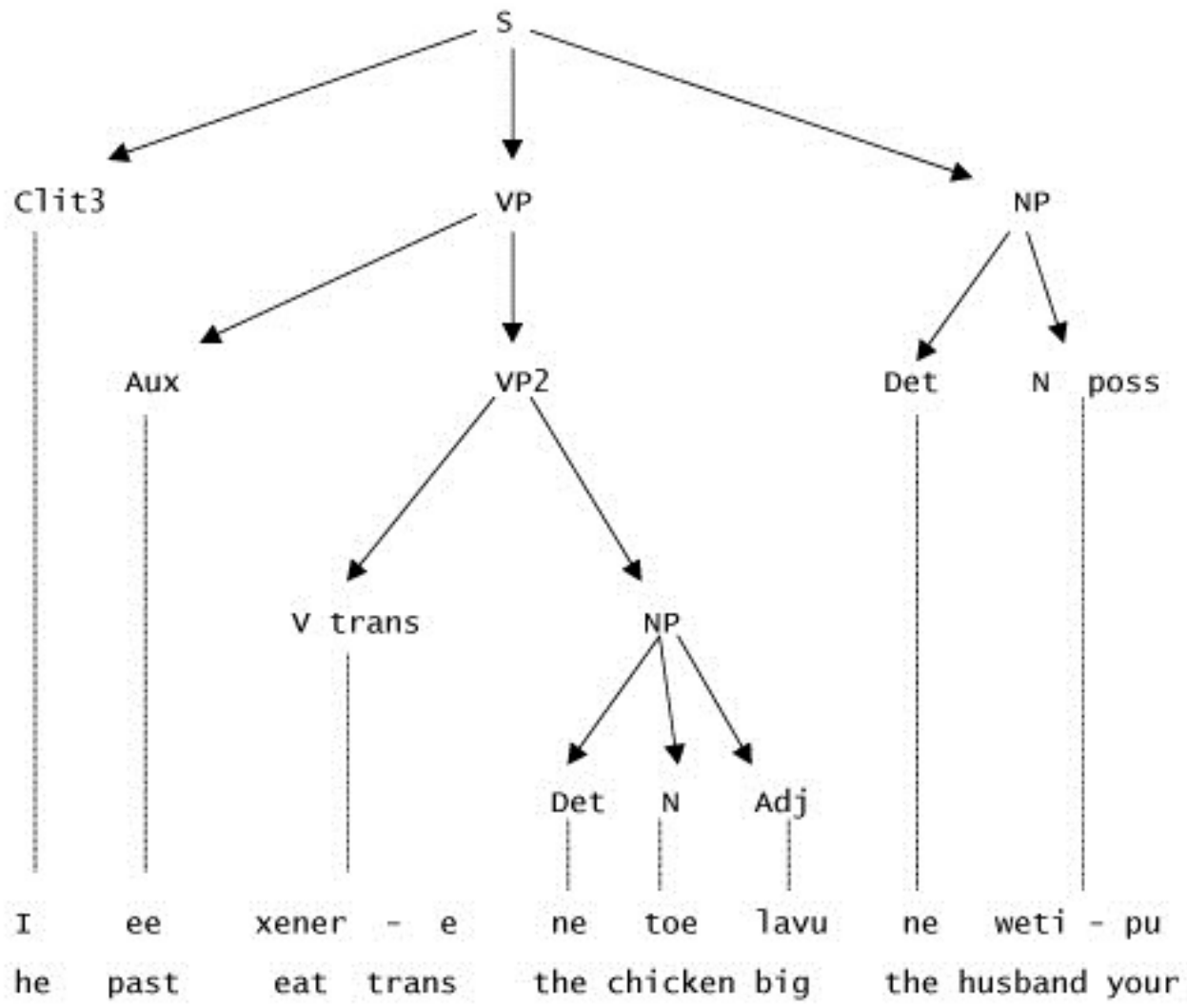
lavu	big	13
loe	black	10

Lexikon,
Produktionsregeln, welche zusammen mit dem Lexikon die Sätze (1) - (13) in Jepeen generierten:

Wort:	Kategorie:
{lezav, xener, xuit}=	V
{lezave, xenere, xuite}=	Vtrans
{lezavi, xeneri}=	Vpassiv
{ee, ne}=	Aux
{ixe, solon, toe, teti, weti}=	N
{tetifu,tetipu,wetifu,wetipu}=	Nposs
{ne}=	Det
{xoieu}=	Pron1
{xoixo}=	Pron2
{xoxie}=	Pron3
{eu}=	Clit1
{xo}=	Clit2
{I}=	Clit3
{lavu, loe}	Adj

Nr.:	Regel:
(R-1)	S -> Clit1 + VP + Pron1
(R-2)	S -> Clit2 + VP + Pron2
(R-3)	S -> Clit3 + VP + Pron3
(R-4)	S -> Clit3 + VP + NP
(R-5)	VP -> Aux + VP2
(R-6)	VP -> VP2
(R-7)	VP2 -> V
(R-8)	VP2 -> V trans + NP
(R-9)	VP2 -> V passiv
(R-10)	NP -> Det + N
(R-11)	NP -> Det + N + Adj
(R-12)	NP -> Det + N poss
(R-13)	NP -> Det + N poss + Adj

Generieren des Satzes in Jepeen, der dem englischen Satz *Your husband ate the big chicken* entspricht:



Lösung 2: Top-down Parser mit Rücksetzen (PT-1)

INPUT TABLE nach der Lexikonphase:

Wörter:	I	han	oine	kennt	kett	die	hot	a	Kent	kett
terminale Kategorien:	**pron1**	**aux1**	**prono**	**verb**	**verb/auxp**	**relpron**	**aux3**	**det**	**n**	**verb/auxp**
Position:	1	2	3	4	5	6	7	8	9	10

WORKING SPACE

A	Derivation	Explanation	P	BACK STORE B	A	P	R
1	S	State at start	1	0			
2	pron1 + aux1 + VP	expansion R-1	1				
3	aux1 + VP	recognized pron1	2				
4	VP	recognized aux1	3				
5	NP2 + VPP	expansion R-2	3	1	4	3	3
6	prono + VPP	expansion R-7	3	2	5	3	8
7	VPP	recognized prono	4				
8	verb	expansion R-4	4	3	7	4	5
9	-	recognized verb	5				
8	verb + auxp	back to A=7 P=4 R=5	4	2			
9	auxp	recognized verb	5				
10	-	recognized auxp	6				

A	Derivation	Explanation	P	B	A	P	R
6	det + n + VPP	back to A=5 P=3 R=8	3	1			
4	VP + RelSatz	back to A=4 P=3 R=3	2	0			
5	NP2 + VPP + RelSatz	expansion R-2	3	1	4	3	3
6	prono + VPP + RelSatz	expansion R-7	3	2	5	3	8
7	VPP + RelSatz	recognized prono	4				
8	verb + RelSatz	expansion R-4	4	3	7	4	5
9	RelSatz	recognized verb	5				
10	relpron + aux3 + VP	expansion R-6	5				
8	verb + auxp + RelSatz	back to A=7 P=4 R=5	4	2			
9	auxp + RelSatz	recognized verb	5				
10	RelSatz	recognized auxp	6				
11	relpron + aux3 + VP	expansion R-6	6				
12	aux3 + VP	recognized relpron	7				
13	VP	recognized aux3	8				
14	NP2 + VPP	expansion R-2	8	2	13	8	3
15	prono + VPP	expansion R-7	8	3	14	8	8
15	det + n + VPP	back to A=14 P=8 R=8	8	2			
16	n + VPP	recognized det	9				
17	VPP	recognized n	10				
18	verb	expansion R-4	10	3	17	10	5
19	-	recognized verb	11				

Parse Tree found (1,3,2,7,5,6,2,8,4)

2. Der Parser würde nicht von alleine aufhören, da die rekursive Regel (R-3) **VP -> VP + Relsatz** immer wieder angewendet werden würde (bis der Speicher überläuft). Deshalb ist auch Reihenfolge der Regeln nicht beliebig. Stünde (R-3) vor (R-2) würde es zu einer Schleife kommen, bevor überhaupt ein Ergebnis gefunden wird.

Lösung 3: Top-down Parser (PT-1) rekursiv programmiert

Trace des Parsers PT-1 mit Stack durch rekursiven Programmaufruf

parser()

Sentence = xo xener xoixo

Preterminals = Clit2+V+ Pron2

Constituents=S

Position=1

Parse=0

expansion(S, Clit2+V+ Pron2, 1, 0)
/* recognize*/
/* assess result */
Success=0
/* expansion */
SaveConstituents=S
SaveParse=0
SavePosition=1
Wanted=S
Loop R=1
Constituents= Clit1 + VP + Pron1
Parse=1

expansion(Clit1 + VP + Pron1, Clit2+V+ Pron2, 1, 1)
/* recognize*/
/* assess result */
Success=0
/* expansion */
SaveConstituents= Clit1 + VP + Pron1
SaveParse=1
SavePosition=1
Wanted=Clit1
Loop R = 1 bis Ende der Regeln
Ende Loop
Return (Success=0)

Loop R=2
Constituents= Clit2 + VP + Pron2
Parse=2

expansion(Clit2 + VP + Pron2, Clit2+V+Pron2, 1, 2, 2)
/* recognize*/
Loop
Constituents= VP + Pron2
Position=2
Ende Loop
/* assess result */
Success=0
/* expansion */
SaveConstituents= VP + Pron2
SaveParse=2
SavePosition=2
Wanted=VP
Loop R = 1 bis R-5
Constituents=Aux + VP2+ Pron2

Parse=2+5

expansion(Aux+VP2+ Pron2, Clit2+V+ Pron2, 2, 2+5)
/* recognize*/
/* assess result */
Success=0
/* expansion */
SaveConstituents=Aux VP2 + Pron2
SaveParse=2+5
SavePosition=2
Wanted=Aux
Loop R = 1 bis Regelende
Return (Success=0)

Loop R = 6
Constituents= VP2+ Pron2
Parse=2+6

expansion(VP2+ Pron2, Clit2+V+ Pron2, 2, 2+6)
/* recognize*/
/* assess result */
Success=0
/* expansion */
SaveConstituents=VP2+ Pron2
SaveParse=2+6
SavePosition=2
Wanted=VP2
Loop R = 1 bis R-7
Constituents=V + Pron2
Parse=2+6+7

expansion(V+ Pron2, Clit2+V+Pron2, 2, 2+6+7)
/* recognize*/

			Loop Constituents=Pron2 Position=3 Constituents= Position=4 Ende Loop **/* assess result */** Position > elements & Constutuents empty Print_parse_tree(2+6+7) Return (Success=1)
		Success=1, break Return (Success=1)	
	Success=1, break Return (Success=1)		
Success=1, break Return (Success=1)			
End Parser			

Der Algorithmus bricht nach einem ersten Ergebnis ab. Um alle möglichen Ergebnisse zu erhalten, muss der break-Befehl entfernt werden, der bewirkt, dass die Schleife durch die Regeln bei Erfolg verlassen wird.

Lösung 4: Top-Down Parser mit paralleler Abarbeitung (PT-2)

EINGABE: *"I ee xenere ne toe lavu ne wetipu"*

WORKING AREA

Pos.	Input	Lexicon	Derivation	Applied rules ("parse")
1	*I*	**Clit3**	S Clit1+VP+Pron1 Clit2+VP+Pron2 Clit3+VP+Pron3 Clit3+VP+NP	 1 2 3 4
			VP+Pron3 VP+NP	3 4
2	*ee*	**Aux**	Aux+VP2+Pron3 Aux+VP2+NP VP2+Pron3 V+Pron3 Vtrans+NP+Pron3 Vpassiv+Pron3 VP2+NP V+NP Vtrans+NP+NP Vpassiv+NP	3,5 4,5 3,6 3,6,7 3,6,8 3,6,9 4,6 4,6,7 4,6,8 4,6,9

			VP2+Pron3 VP2+NP	3,5 4,5
3	*xenere*	**Vtrans**	V+Pron3 Vtrans+NP+Pron3 Vpassiv+Pron3 V+NP Vtrans+NP+NP Vpassiv+NP	3,5,7 3,5,8 3,5,9 4,5,7 4,5,8 4,5,9
			NP+Pron3 NP+NP	3,5,8 4,5,8
4	*ne*	**Det**	Det+N+Pron3 Det+N+Adj+Pron3 Det+Nposs+Pron3 Det+Nposs+Adj+Pron3 Det+N+NP Det+N+Adj+NP Det+Nposs+NP Det+Nposs+Adj+NP	3,5,8,10 3,5,8,11 3,5,8,12 3,5,8,13 4,5,8,10 4,5,8,11 4,5,8,12 4,5,8,13
			N+Pron3 N+Adj+Pron3 Nposs+Pron3 Nposs+Adj+Pron3 N+NP N+Adj+NP Nposs+NP Nposs+Adj+NP	3,5,8,10 3,5,8,11 3,5,8,12 3,5,8,13 4,5,8,10 4,5,8,11 4,5,8,12 4,5,8,13

5	*toe*	**N**	Pron3 Adj+Pron3 NP Adj+NP	3,5,8,10 3,5,8,11 4,5,8,10 4,5,8,11
6	*lavu*	**Adj**	Pron3 Adj+Pron3 Det+N Det+N+Adj Det+Nposs Det+Nposs+Adj Adj+NP	3,5,8,10 3,5,8,11 4,5,8,10,10 4,5,8,10,11 4,5,8,10,12 4,5,8,10,13 4,5,8,11
			Pron3 NP	3,5,8,11 4,5,8,11
7	*ne*	**Det**	Pron3 Det+N Det+N+Adj Det+Nposs Det+Nposs+Adj	3,5,8,11 4,5,8,11,10 4,5,8,11,11 4,5,8,11,12 4,5,8,11,13
			N N+Adj Nposs Nposs+Adj	4,5,8,11,10 4,5,8,11,11 4,5,8,11,12 4,5,8,11,13
8	*wetipu*	**Nposs**	_ Adj	**4,5,8,11,12** 4,5,8,11,13

Der Parse lautet 4,5,8,11,12

Lösung 5: Top-down Parser mit Greibach-Nomalform (PT-3)

Grammatik des Schwäbischen:

Regeln:		**Lexikon:**	***Hochdeutsch:***
(R-1)	S -> pron1 + aux1 + VP	aux1 = *han*	*habe*
(R-2)	VP -> NP2 + VPP	aux3 = *hot*	*hat*
(R-3)	VP -> VP + RelSatz	auxp = *kett*	*gehabt*
(R-4)	VPP -> verb	det = *a*	*ein*
(R-5)	VPP -> verb + auxp	n = *Kent*	*Kind*
(R-6)	Relsatz -> relpron + aux3 + VP	pron1 = *i*	*ich*
(R-7)	NP2 -> prono	prono = *oine*	*eine (Frau)*
(R-8)	NP2 -> det + n	relpron = *die*	*die*
		verb = {*kennt, kett*}	*gekannt, gehabt*

Problem: Die rekursive Regel (R-3) **VP -> VP + RelSatz** würde bei der Expansion zu einer unendlichen Schleife führen.

Lösung: Man schreibt die Regel (R-3) um in **VP -> NP2 + VPP + Relsatz**
Das ist sowieso besser, weil der Relativsatz ja von der NP2 abhängt..

Grammatik des Schwäbischen in Greibach-Normalform:

(R-1)	(S, pron1)	→	Aux1VP
(R-2)	(VP , det)	→	N + VPP
(R-3)	(VP , det)	→	N + VPP+ RelSatz
(R-4)	(VP , prono)	→	VPP
(R-5)	(VP , prono)	→	VPP+ RelSatz
(R-6)	(VPP , verb)	→	-
(R-7)	(VPP , verb)	→	Auxp
(R-8)	(Aux1VP , aux1)	→	VP
(R-9)	(Aux3VP , aux3)	→	VP
(R-10)	(Auxp , auxp)	→	-
(R-11)	(Relsatz , relpron)	→	Aux3VP
(R-12)	(N , n)	→	-

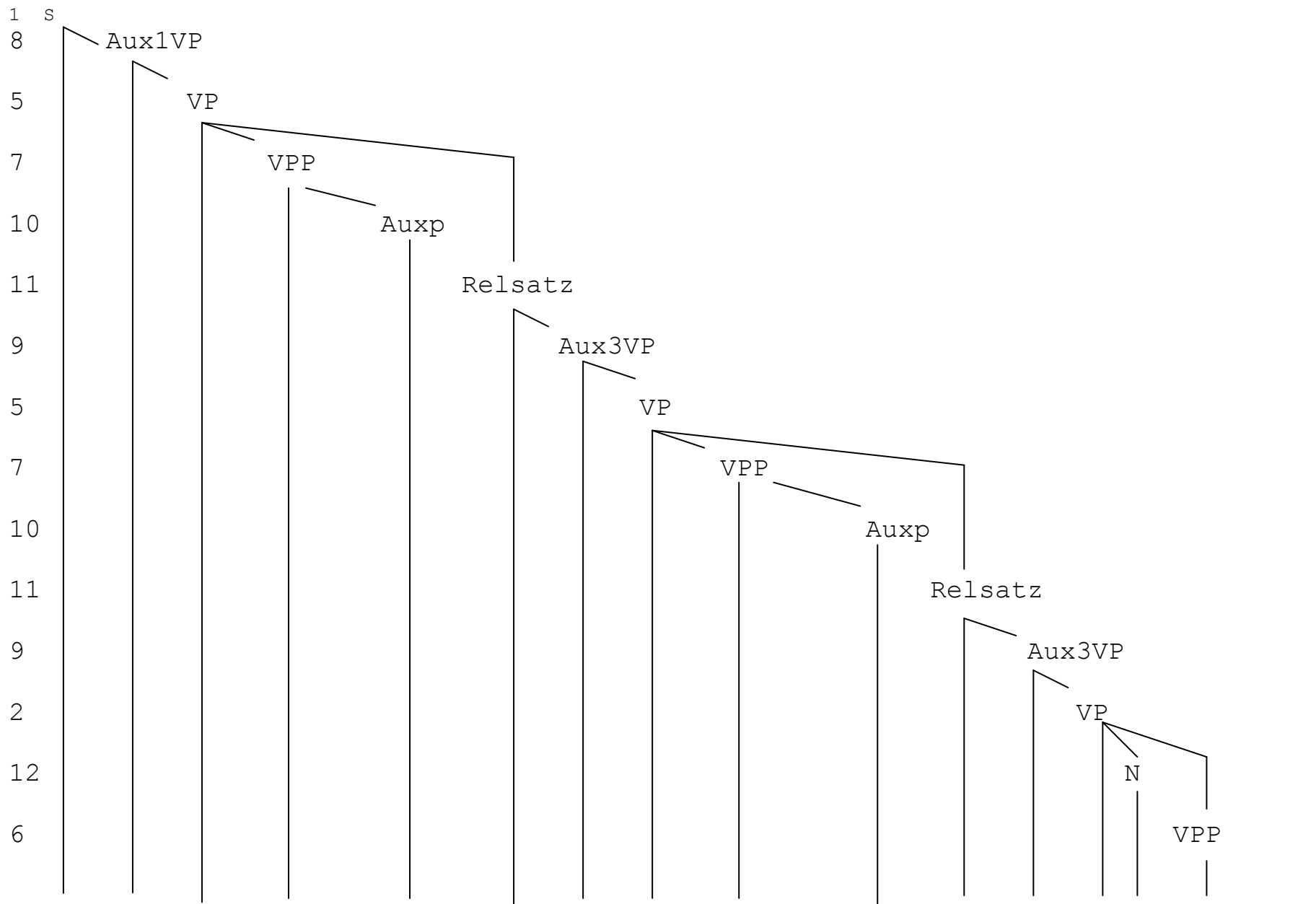
EINGABE: "I han oine kennt kett die hot oine kennt kett die hot a Kent kett"

WORKING AREA

Pos.	Input	Derivation	Preterminal	Prediction	Applied rules ("parse")
1	**I**	S	**pron1**	Aux1VP	1
2	**han**	Aux1VP	**aux1**	VP	1,8
3	**oine**	VP	**prono**	VPP VPP+ RelSatz	1,8,4 1,8,5
4	**kennt**	VPP VPP+ RelSatz	**verb**	- Auxp RelSatz Auxp + Relsatz	1,8,4,6 1,8,4,7 1,8,5,6 1,8,5,7
5	**kett**	Auxp RelSatz Auxp + Relsatz	**verb/auxp**	- Relsatz	1,8,4,7,10 1,8,5,7,10
6	**die**	Relsatz	**relpron**	Aux3VP	1,8,5,7,10,11
7	**hot**	Aux3VP	**aux3**	VP	1,8,5,7,10,11,9
8	**oine**	**VP**	**prono**	VPP VPP+ RelSatz	1,8,5,7,10,11,9,4 1,8,5,7,10,11,9,5
9	**kennt**	VPP VPP+ RelSatz	**verb**	- Auxp RelSatz Auxp + Relsatz	1,8,5,7,10,11,9,4,6 1,8,5,7,10,11,9,4,7 1,8,5,7,10,11,9,5,6 1,8,5,7,10,11,9,5,7

10	**kett**	Auxp RelSatz Auxp + Relsatz	**verb/auxp**	- Relsatz	1,8,5,7,10,11,9,4,7,10 1,8,5,7,10,11,9,5,7,10
11	**die**	Relsatz	**relpron**	Aux3VP	1,8,5,7,10,11,9,5,7,10,11
12	**hot**	Aux3VP	**aux3**	VP	1,8,5,7,10,11,9,5,7,10,11,9
13	**a**	VP	**det**	N + VPP N + VPP+ RelSatz	1,8,5,7,10,11,9,5,7,10,11,9,2 1,8,5,7,10,11,9,5,7,10,11,9,3
14	**Kent**	N + VPP N + VPP+ RelSatz	**n**	VPP VPP+ RelSatz	1,8,5,7,10,11,9,5,7,10,11,9,2,12 1,8,5,7,10,11,9,5,7,10,11,9,3,12
15	**kett**	VPP VPP+ RelSatz	**verb/auxp**	- Auxp RelSatz Auxp + Relsatz	1,8,5,7,10,11,9,5,7,10,11,9,2,12,6 1,8,5,7,10,11,9,5,7,10,11,9,2,12.7 1,8,5,7,10,11,9,5,7,10,11,9,3,12,6 1,8,5,7,10,11,9,5,7,10,11,9,3,12,7

Parse: 1,8,5,7,10,11,9,5,7,10,11,9,2,12,6


1 S
8 Aux1VP
5 VP
7 VPP
10 Auxp
11 Relsatz
9 Aux3VP
5 VP
7 VPP
10 Auxp
11 Relsatz
9 Aux3VP
2 VP
12 N
6 VPP
pron1 aux1 prono verb verb/auxp relpron aux3 prono verb verb/auxp relpron aux3 det n verb/auxp
I han oine kennt kett die hot oine kennt kett die hot a Kent kett

Lösung 6: Top-Down Chart Parser nach Earley (PT-4)

Regeln	
(R-1)	**S -> NP VV**
(R-2)	**S -> NP VA**
(R-3)	**VV -> vi**
(R-4)	**VV -> vt NP**
(R-5)	**VA -> aux Vinf**
(R-6)	**Vinf -> Zuvi**
(R-7)	**Vinf -> NP Zuvt**
(R-8)	**Zuvi -> zu vi**
(R-9)	**Zuvt -> zu vt**
(R-10)	**NP -> pron**
(R-11)	**NP -> det n**
(R-12)	**NP -> adv det n**

Lexikon		
vi	=	*{singen, labern, schweigen}*
vt	=	*{haben, trinken, bezahlen}*
aux	=	*{haben, scheinen}*
zu	=	*{zu}*
det	=	*{die, meine, keine, drei}*
adv	=	*{noch, schon, erst}*
n	=	*{Schnäpse, Bouletten, Kekse}*
pron	=	*{wir, sie, manche, alle}*

1. Eingabe:

wir	haben	noch	drei	Bouletten
pron	aux/vt	adv	det	n
0	1	2	3	4

Section 0:

	Divided productions	Left margin	Right margin	Explanation
(1)	**# -> .S**	0	0	Starting state
(2)	**S -> .NP VV**	0	0	predictor for (1) by R-1
(3)	**S -> .NP VA**	0	0	predictor for (1) by R-2
(4)	**NP -> .pron**	0	0	predictor for (2) by R-10
(5)	**NP -> .det n**	0	0	predictor for (2) by R-11
(6)	**NP -> .adv det n**	0	0	predictor for (2) by R-12
(*)	**NP -> .pron**	0	0	predictor for (3) by R-10
(*)	**NP -> .det n**	0	0	predictor for (3) by R-11
(*)	**NP -> .adv det n**	0	0	predictor for (3) by R-12

Section 1:

(7)	**NP -> pron.**	0	1	scanner for (4), wir
(8)	**S -> NP . VV**	0	1	completor (7) in (2)
(9)	**S -> NP . VA**	0	1	completor (7) in (3)
(10)	**VV -> .vi**	1	1	predictor for (8) by R-3
(11)	**VV -> .vt NP**	1	1	predictor for (8) by R-4
(12)	**VA -> .aux Vinf**	1	1	predictor for (9) by R-5

Section 2:

(13)	**VV -> vt . NP**	1	2	scanner for (11), haben
(14)	**VA -> aux . Vinf**	1	2	scanner for (12), haben
(15)	**NP -> .pron**	2	2	predictor for (13) by R-10
(16)	**NP -> .det n**	2	2	predictor for (13) by R-11
(17)	**NP -> .adv det n**	2	2	predictor for (13) by R-12
(18)	**Vinf -> .Zuvi**	2	2	predictor for (14) by R-6
(19)	**Vinf -> .NP Zuvt**	2	2	predictor for (14) by R-7

Section 3:

(20)	**NP -> adv . det n**	2	3	scanner for (17), noch

wieder *Section 2:*

(21)	**Zuvi -> zu vi**	2	2	predictor for (18) by R-8
(*)	**NP -> pron**	2	2	predictor for (19) by R-10
(*)	**NP -> det n**	2	2	predictor for (19) by R-11
(*)	**NP -> adv det n**	2	2	predictor for (19) by R-12

Section 4:

(25)	**NP -> adv det . n**	2	4	scanner for (20), drei

Section 5:

(26)	**NP -> adv det n.**	2	5	scanner for (25), Bouletten
(27)	**VV -> vt NP.**	1	5	completor (26) in (13)
(28)	**Vinf -> NP . Zuvt**	2	5	completor (26) in (19)
(29)	**S -> NP VV.**	0	5	completor (27) in (8)
(30)	**# -> S.**	0	5	completor (29) in (1)

Zeilen mit (*) werden in Wirklichkeit nicht erzeugt, bzw. sofort wieder getilgt, weil dieselbe Derivation schon vorhanden ist.

2. Anschliessend war der folgende Satz zu parsen:

wir	haben	noch	drei	Bouletten	zu	bezahlen
pron	aux/vt	adv	det	n	zu	vt
0	1	2	3	4	5	6

Es kommen folgende Zeilen hinzu:

Section 5:

(31)	**Zuvt -> .zu vt**	5	5	predictor for (28) by R-9

Section 6:

(32)	**Zuvt -> zu . vt**	5	6	scanner for (31), zu

Section 7:

(33)	**Zuvt -> zu vt.**	5	7	scanner for (32), bezahlen
(34)	**Vinf -> NP Zuvt.**	2	7	completor (33) in (28)
(35)	**VA -> aux Vinf.**	1	7	completor (34) in (14)
(36)	**S -> NP VA.**	0	7	completor (35) in (9)
(37)	**# -> S.**	0	7	completor (36) in (1)

3. Bei einem Parser mit Backtracking wie PT-1 würden die Zeilen (4), (5), (6), (7), (15), (16), (17), (20), (25), (26) zweimal erzeugt, statt sie wie bei PT-4 wiederzuverwenden.

4. Zu einer Unregelmäßigkeit bei der aufsteigenden Ordnung der rechten Grenzen (welche die Sections definieren) kommt es, wenn der Scanner den Predictor überholt. Das ist potentiell dann der Fall, wenn Regeln mit terminalen Symbolen am Anfang vor solchen mit nicht-terminalen an derselben Position anwendbar sind.

Lösung 7: Bottom-Up Chart Parser nach Cocke (PT-5)

2. Grammatik des Schwäbischen in Chomsky-Normalform:

Regeln:		**Lexikon:**	***Hochdeutsch:***
(R-1)	S -> pron1 + V1	aux1 = *han*	*habe*
(R-2)	V1 -> aux1 + Vo	aux3 = *hot*	*hat*
(R-3)	V1 -> aux1 + Vp	auxp = *kett*	*gehabt*
(R-4)	V1-> aux1 + Vr	det = *a*	*ein*
(R-5)	Vo -> prono + verb	n = *Kent*	*Kind*
(R-6)	Vo -> No + verb	pron1 = *i*	*ich*
(R-7)	No -> det + n	prono = *oine*	*eine (Frau)*
(R-8)	Vp -> Vo + auxp	relpron = *die*	*die*
(R-9)	Vr -> Vp + RelSatz *)	verb = {*kennt, kett*}	*gekannt, gehabt*
(R-10)	Vr -> Vo + Relsatz *)		
(R-11)	Relsatz -> relpron + V3		
(R-12)	V3 -> aux3 + Vo		
(R-13)	V3 -> aux3 + Vp		
(R-14)	V3-> aux3 + Vr		

Die Subklassifikation von V und aux bedeutet: 1=erste Person, 3=dritte Person, p=plus-quam-perfekt, o=mit Objekt, r=mit Relativsatz - solche Merkmale könnte man natürlich besser mit komplexen Kategorien darstellen.

*) Der Anschluss des Relativsatzes an Vp oder Vo ist linguistisch nicht korrekt. Der richtige Anschluss an No ist aber diskontinuierlich, was die kontextfreie Phrasenstrukturgrammatik nicht darstellen kann.

2. Bottom-Up Chart Parsing

INPUT TABLE

Input:	*I*	*han*	*oine*	*kennt*	*kett*	*die*	*hot*	*a*	*Kent*	*kett*
Lexicon:	**pron1**	**aux1**	**prono**	**verb**	**verb/auxp**	**relpron**	**aux3**	**det**	**n**	**verb/auxp**
Margins:	0 1	1 2	2 3	3 4	4 5	5 6	6 7	7 8	8 9	9 10

WORKING TABLE

*) Head blue

Section 1:

	Category	Left margin	Right margin	Left IC	Right *) IC	Explanation
(1)	**pron1**	0	1	-	-	shift *I*

Section 2:

(2)	**aux1**	1	2	-	-	shift *han*

Section 3:

(3)	**prono**	2	3	-	-	shift *oine*

Section 4:

(4)	**verb**	3	4	-	-	shift *kennt*
(5)	**Vo**	2	4	3	**4**	reduce by R-5
(6)	**V1**	1	4	**2**	5	reduce by R-2
(7)	**S**	0	4	1	**6**	reduce by R-1

Section 5:

(8)	**verb**	4	5	-	-	shift *kett*
(9)	**auxp**	4	5	-	-	shift *kett*
(10)	**Vp**	2	5	5	**9**	reduce by R-8
(11)	**V1**	1	5	**2**	10	reduce by R-3
(12)	**S**	0	5	1	**11**	reduce by R-1

Section 6:

(13)	**relpron**	5	6	-	-	shift *die*

Section 7:

(14)	**aux3**	6	7	-	-	shift *hot*

Section 8:

(15)	**det**	7	8	-	-	shift *a*

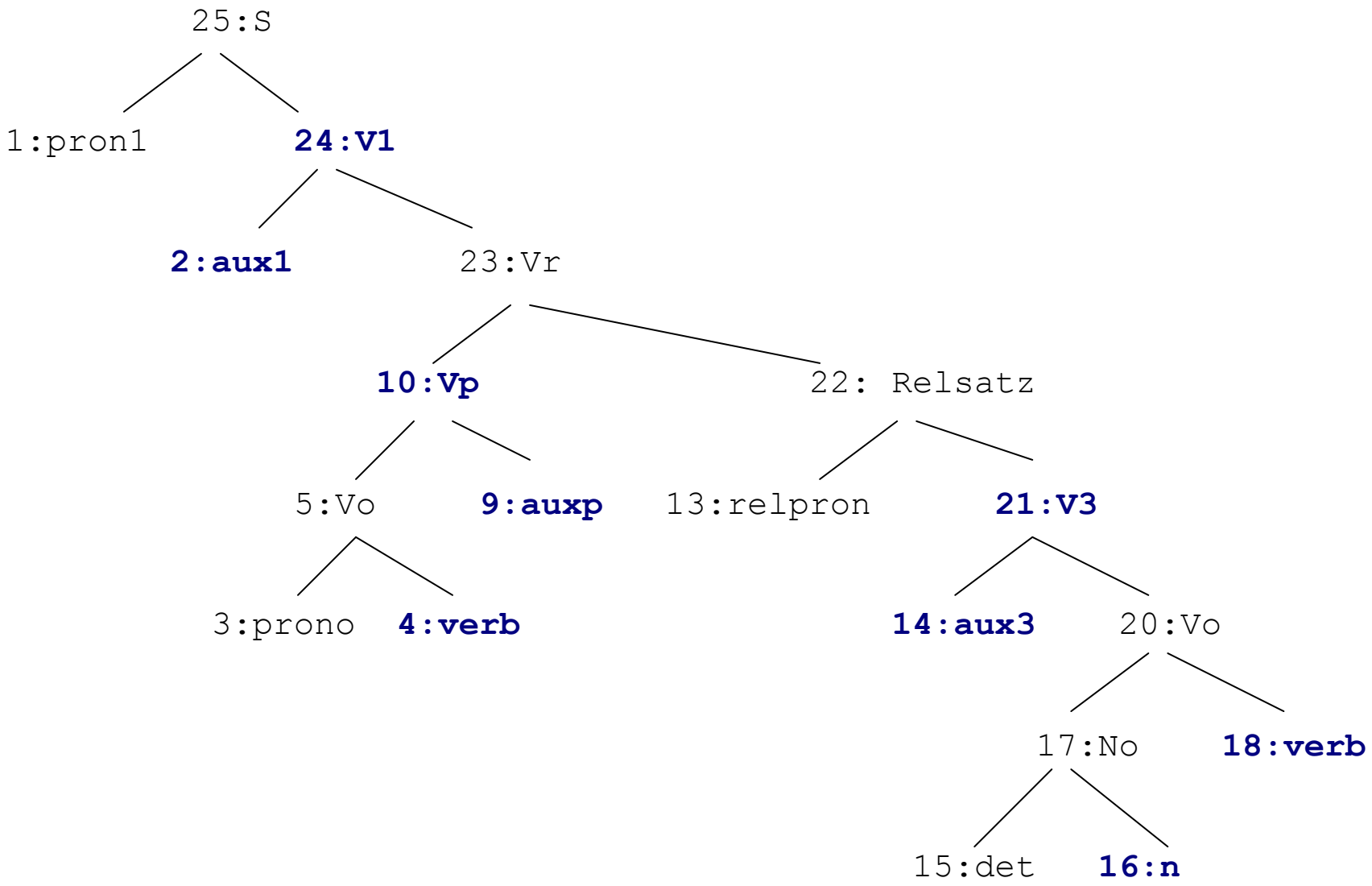
Section 9:

(16)	**n**	8	9	-	-	shift *Kent*
(17)	**No**	7	9	15	**16**	reduce by R-7

Section 10:

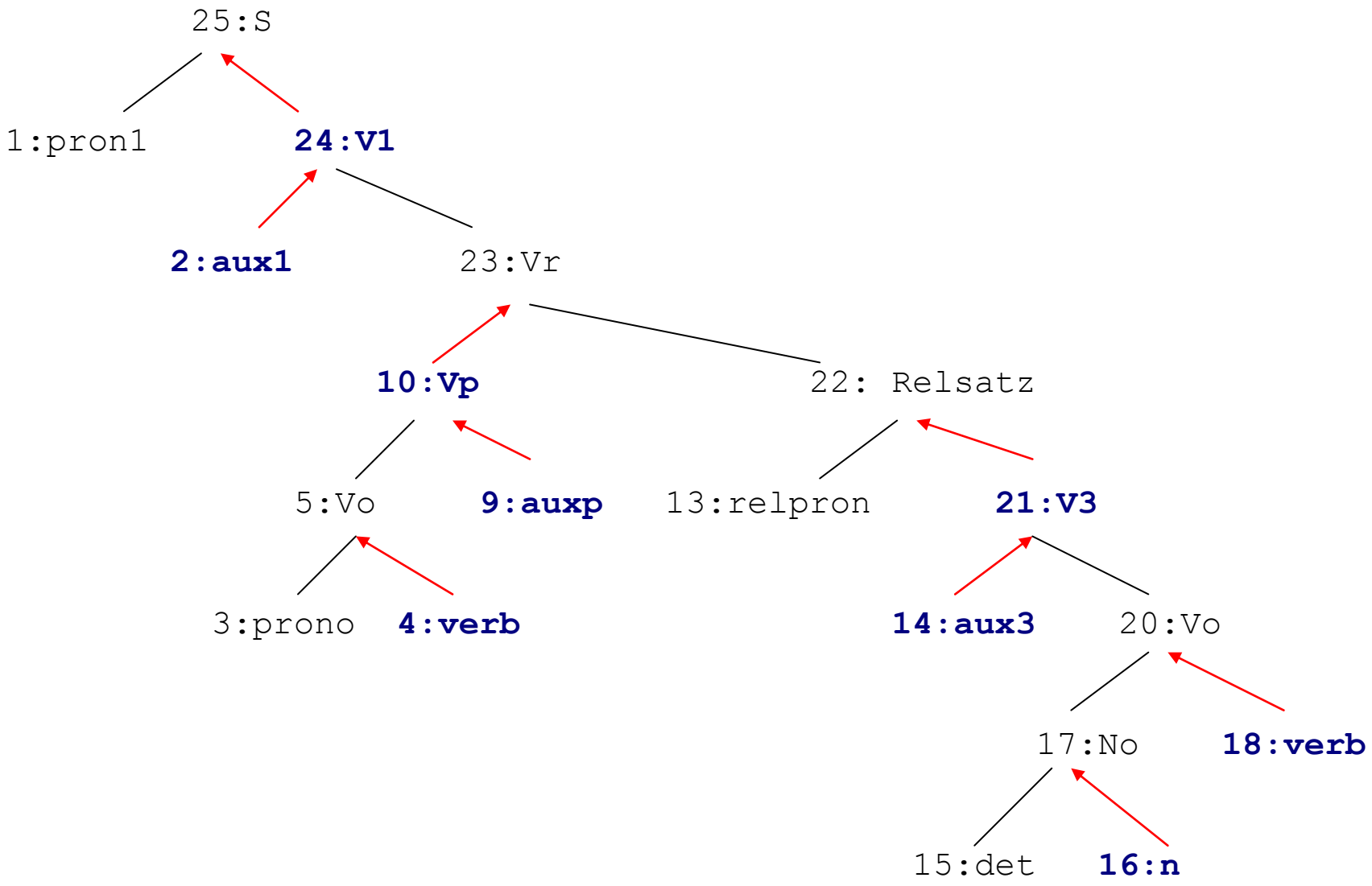
(18)	**verb**	9	10	-	-	shift *kett*
(19)	**auxp**	9	10	-	-	shift *kett*
(20)	**Vo**	7	10	17	**18**	reduce by R-6
(21)	**V3**	6	10	**14**	20	reduce by R-12
(22)	**Relsatz**	5	10	13	**21**	redcuce by R-11
(23)	**Vr**	2	10	**10**	22	reduce by R-9
(24)	**V1**	1	10	**2**	23	recuce by R-4
(25)	**S**	0	10	1	**24**	recuce by R-1

3. Konstituenzbaum nach der Working Table (Heads fettgedruckt)

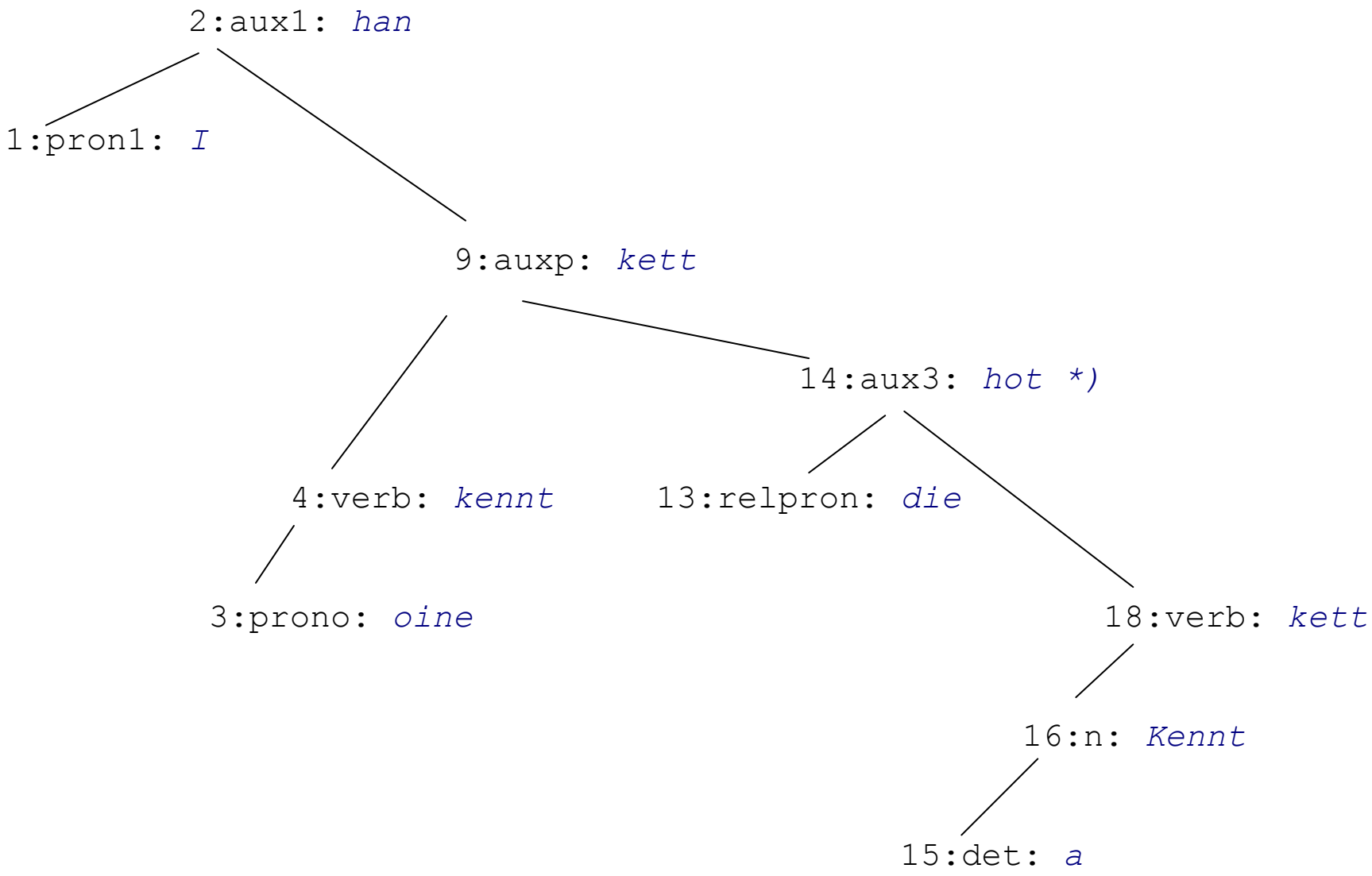


Konstruktion eines Dependenzbaums

Der jeweilige Mutterknoten wird ersetzt durch den Head unter den beiden IC-Knoten. Die Heads werden also nach oben hin verschoben, wodurch ein Dependenzbaum entsteht:



Ergebnis:



*) Müsste eigentlich von *oine* abhängen. Siehe Bemerkung oben.

Lösung 8: Chart-Parsing mit einer Kategorialgrammatik

Lexikon

i	=	pron1
die	=	pron3
han	=	(pron1\S) / Vo, (pron1\S) /Vp
hot	=	(pron3\S) / Vo, (pron3\S) /Vp
kett	=	Vo\Vp
oine	=	prono
a	=	No/n
Kent	=	n
kennt, kett}	=	No\Vo, prono\Vo

Input:	***die***	***hot***	***a***	***Kent***	***kett***
Lexicon:	pron3	(pron3\S) / Vo (pron3\S) /Vp	No/n	n	Vo\Vp No\Vo prono\Vo
Position:	0 1	1 2	2 3	3 4	4 5

WORKING TABLE

Section 1:

	Category	Left margin	Right margin	Left IC	Right IC	Explanation
(1)	pron3	0	1	-	-	scanner *die*

Section 2:

(2)	(pron3\S) / Vo	1	2	-	-	scanner *hot*
(3)	(pron3\S) /Vp	1	2	-	-	scanner *hot*

Section 3:

(4)	No/n	2	3	-	-	scanner *a*

Section 4:

(5)	n	3	4	-	-	scanner *Kent*
(6)	No	2	4	4	5	completed 4 by 5

Section 5:

(7)	Vo\Vp	4	5	-	-	scanner *kett*
(8)	No\Vo	4	5	-	-	scanner *kett*
(9)	prono\Vo	4	5	-	-	scanner *kett*
(10)	Vo	2	5	6	8	completed 8 by 6
(11)	pron3\S	1	5	2	10	completed 11 by 1
(11)	S	0	5	1	11	

Lösung 9: Tabellengesteuerter Shift-Reduce Parser (PT-6)

1. Herstellung der Tabellen
Zerlegung der Regeln in Zustände

Zustände	Divided productions	Weitere Expansion	Regel
(z-0)	S' -> .S	S -> . pron1 aux1 VP	
(z-1)	S' -> S.		acc
(z-2)	S -> pron1 . aux1 VP		
(z-3)	S -> pron1 aux1 . VP	VP -> . NP2 VPP VP -> . VP RelSatz NP2 -> . prono NP2 -> . det n	
(z-4)	S -> pron1 aux1 VP.		re 1
(z-5)	VP -> NP2 . VPP	VPP -> .verb VPP -> .verb auxp	
(z-6)	VP -> NP2 VPP .		re 2
(z-7)	VP -> VP . RelSatz	Relsatz -> . relpron aux3 VP	
(z-8)	VP -> VP RelSatz.		re 3
(z-9)	VPP -> verb.		re 4

(z-10)	VPP -> verb . auxp		
(z-11)	VPP -> verb auxp.		re 5
(z-12)	Relsatz -> relpron . aux3 VP		
(z-13)	Relsatz -> relpron aux3 . VP	VP -> . NP2 VPP VP -> . VP RelSatz NP2 -> . prono NP2 -> . det n	
(z-14)	Relsatz -> relpron aux3 VP.		re 6
(z-15)	NP2 -> prono.		re 7
(z-16)	NP2 -> det . n		
(z-17)	NP2 -> det n.		re 8

Funktion FOLLOW

```
FOLLOW(S)  = $
FOLLOW(VP) = FOLLOW(S) = $
FOLLOW(VP) = RelSatz
FOLLOW(VP) = FOLLOW(RelSatz)
FOLLOW(NP2) = VPP
FOLLOW(VPP) = FOLLOW(VP)= $, Relsatz, FOLLOW(RelSatz)
FOLLOW(Relsatz) = FOLLOW(VP)= $, Relsatz, FOLLOW(RelSatz)
```

Funktion FIRST

```
FIRST($)   = $
FIRST(NP2) =  prono, det
FIRST(VP)  = FIRST(NP2), FIRST(VP) = prono,det
FIRST(VPP)  = verb
FIRST(RelSatz) = relpron
```

ACTION TABLE

z	aux1	aux3	auxp	det	n	pron1	prono	relpron	verb	$
0						sh2				
1										acc
2	sh3									
3				sh16			sh15			
4										re1
5									sh9, sh10	
6								re2		re2
7								sh12		
8								re3		re3
9								re4		re4
10			sh11							
11								re5		re5
12		sh13								
13				sh16			sh15			
14								re6		re6
15									re7	
16					sh17					
17									re8	

GOTO TABLE

NP2	VP	VPP	Relsatz	S
				1
5	4,7			
		6		
			8	
5	7,14			

2. Test

Input:	*I*	*han*	*oine*	*kennt*	*die*	*hot*	*a*	*Kent*	*kett*
Lexicon:	**pron1**	**aux1**	**prono**	**verb**	**relpron**	**aux3**	**det**	**n**	**auxp**
Position:	1	2	3	4	5	6	7	8	9

P:	**States and Descriptions:**	**Explanation:**
0	**0**	starting state
1	**0 -pron1- 2**	sh2
2	**0 -pron1- 2 -aux1- 3**	sh3
3	**0 -pron1- 2 -aux1- 3 -prono- 15**	sh15
3	**0 -pron1- 2 -aux1- 3 -NP2(prono)- 5**	re7, goto5
4	**0 -pron1- 2 -aux1- 3 -NP2(prono)- 5 -verb-|** **9** / **10**	sh9 or sh10 10 fails
4	**0 -pron1- 2 -aux1- 3 -NP2(prono)- 5 -VPP(verb)- 6**	re4, goto6
4	**0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-|** **4** / **7**	re2, goto4 or 7 4 fails

```
5   0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12

6   0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12 -aux3- 13

7   0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12 -aux3- 13
                                                  13 -det- 16

8   0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12 -aux3- 13
                                                  13 -det- 16 -n- 17

8   0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12 -aux3- 13
                                                  13 -NP2(det n)- 5

9   0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12 -aux3- 13
                                                            9
                                                  13 -NP2(det n)- 5 -verb-|
                                                                       10  -fails

9   0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12 -aux3- 13
                                                  13 -NP2(det n)- 5 -VPP(verb)-6
```

9 **0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7 -relpron- 12 -aux3- 13**

7 fails

13 -VP(NP2(det n)VPP(verb)-|

14

9 **0 -pron1- 2 -aux1- 3 -VP(NP2(prono)VPP(verb)-7**

7 -Relsatz(relpron) (aux3) (VP(NP2(det n)VPP(verb)) -8

9 **0 -pron1- 2 -aux1- 3 -VP(VP(NP2(prono)VPP(verb))(Relsatz(relpron)**
(aux3) (VP(NP2(det n)VPP(verb)) -4,7

9 **0 S(pron1)(aux1)(VP(VP(NP2(prono)VPP(verb))(Relsatz(relpron)**
(aux3) (VP(NP2(det n)VPP(verb)) 1

acc

Lösung 10: Deterministischer FTN-Parser für reguläre Ausdrücke (PT-7)

Ein Beispiel für einen Grammatikformalismus ist z.B. G1 aus dem Skript::

Rules	
(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 -> prep + NP

Lexicon		
vi	=	{sleep, fish}
vt	=	{study, visit, see, enjoy}
det	=	{the, no, my, many}
adj	=	{foreign, beautiful}
n	=	{tourists, pyramids, friends, fish, cans, Egypt, we, they}
prep	=	{in, by, with}

1. Beschreibung des Formalismus

Jede Regel besteht aus einer nicht-terminalen Kategorie, einem Pfeil und einer oder mehrerer terminaler oder nicht-terminaler Kategorien, die durch "+" getrennt sind. Nicht-terminale Kategorien bestehen aus Großbuchstaben, terminale Kategorien aus Kleinbuchstaben.

Ein Lexikoneintrag besteht aus einer terminalen Kategorie, einem Gleichheitszeichen, einer öffnenden eckigen Klammer, einer oder mehrerer Wörter, die durch ein Komma getrennt sind, und einer schließenden eckigen Klammer. Wörter bestehen aus einem Großbuchstaben gefolgt von Kleinbuchstaben oder nur aus Kleinbuchstaben.

2. Reguläre Ausdrücke für diesen Formalismus

Kategorien:

U = {"A" |"B" | "C" .... | "Z"}

L = {"a" | "b" | "c" .... | "z"}

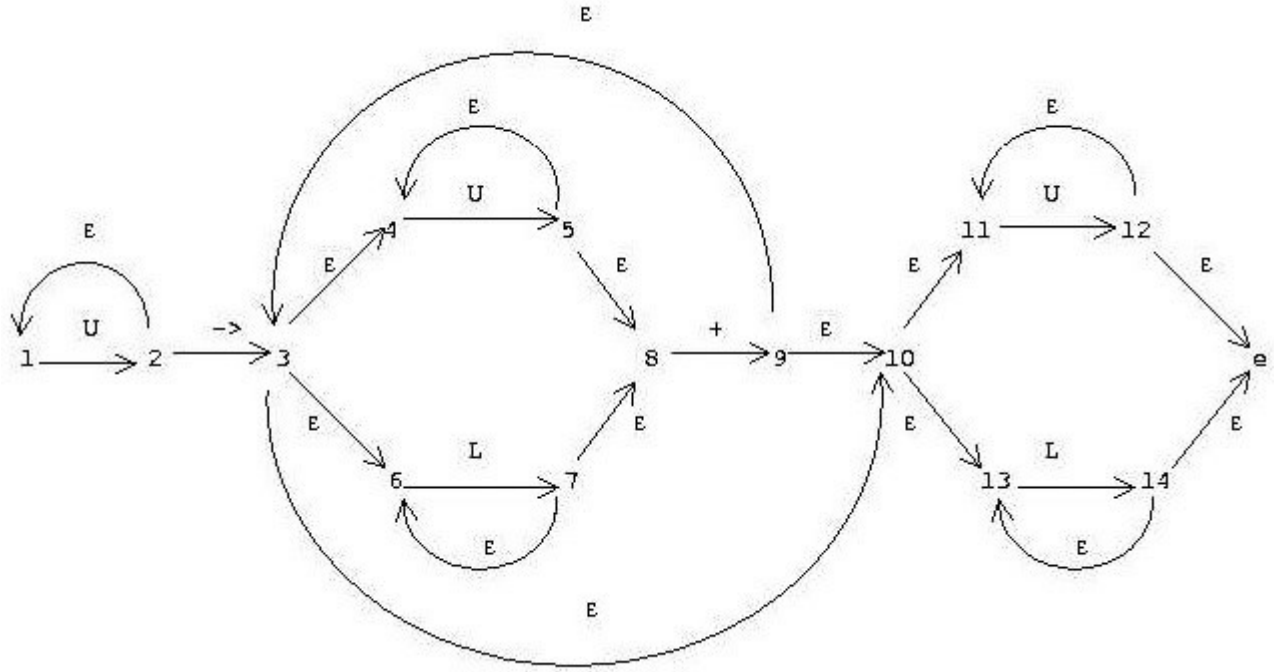
Regeln:

U^+ "->" {(U^+ | L^+) "+" }* (U^+ | L^+)

Lexikoneinträge:

L^+ "=" "{" (U? L^+ ",")* U? L^+ "}"

3. Nicht-deterministisches endliches Übergangsnetzwerk zum regulären Ausdruck für die Regeln.



4. Übergangstabelle für einen deterministischen endlichen Erkenner.

Zustand	U	L	->	+
1	2/1			
2/1	2/1		3/4/6/9/10/13	
3/4/6/10/11/13	5/4/8/12/11/e	7/6/8/14/13/e		
5/4/8/12/11/e	5/4/8/12/11/e			9/3/4/6/10/11/13
7/6/8/14/13/e		7/6/8/14/13/e		9/3/4/6/10/11/13
9/3/4/6/10/11/13	5/4/8/12/11/e	7/6/8/14/13/e		

5. Demonstration des Parsers anhand einer Regel der Grammatik

Eingabe: VP -> vt + NP + PP

Ausgangszustand:	Position:	Eingabesymbol:	Zielzustand:
1	1	V	2/1
2/1	2	P	2/1
2/1	3	->	3/4/6/9/10/13
3/4/6/9/10/13	4	v	7/6/8/14/13/e
7/6/8/14/13/e	5	t	7/6/8/14/13/e
7/6/8/14/13/e	6	+	9/3/4/6/10/11/13
9/3/4/6/10/11/13	7	N	5/4/8/12/11/e
5/4/8/12/11/e	8	P	5/4/8/12/11/e
5/4/8/12/11/e	9	+	9/3/4/6/10/11/13
9/3/4/6/10/11/13	10	P	5/4/8/12/11/e
5/4/8/12/11/e	11	P	5/4/8/12/11/e

Lösung 11: Augmented Transition Network (ATN) Parser (PT-8)

Analyse:

Römische Zahlen bestehen aus Buchstaben, die bestimmte Werte haben, und zwar
I=1, V=5, X=10, L=50, C=100, D=500, M=1000

In der Regel folgen die Buchstaben in der Reihenfolge gleicher oder abnehmende Werte. Ausnahme: Ein einzelner Buchstabe mit kleinerem Wert kann jedem Buchstaben vorausgehen. I,X,C.M treten maximal dreimal hintereinander auf, V, L, D überhaupt nur einmal.

Der Gesamtwert einer römischen Zahl ergibt sich durch Addition und Subtraktion der Werte für die einzelnen Buchstaben. Dabei wird der Wert eines Buchstabens von dem Wert des nächsten Buchstabens subtrahiert, wenn er kleiner ist als letzterer, sonst werden die Werte addiert.

Algorithmus:

Für jeden Buchstaben gibt es eine Kante. Alle Kanten führen aber zum selben Zustand zurück, bis ein Leerzeichen das Ende der römischen Zahl anzeigt. Sämtliche Regeln zur Kombinatorik werden durch Bedingungen an den Kanten implementiert.

1. ATN-GRAMMATIK FÜR RÖMISCHE ZAHLEN
(*Kommentare kursiv*)

***node* PUSH RomanNumber**

***arc* JUMP**

SETR last 0
Dieses Register enthält den Wert des letzten röm. Buchstabens.
SETR count 0
Dieses Register enthält die Anzahl der Vorkommen desselben Buchstabens
SETR sum 0
Dieses Register enthält den Zahlenwert, der als arabische Zahl ausgegeben wird.
TO NextCharacter

***node* NextCharacter**
Es wird immer der aktuelle mit dem vorangehenden Buchstaben verglichen;
es wird immer der vorangegangene Buchstabens zum Register sum *addiert oder subtrahiert, nicht der aktuelle.*

***arc 1* WRD I**

```
if <GETR last> = 1 & <GETR count> =3 then false
else if <GETR last> = 1
        then {SETR sum  <GETR sum + 1>
        <SETR count <GETR count + 1>}
else  {SETR sum  <GETR sum + GETR last>
        SETR last  1
        SETR count 1}
TO  NextCharacter
```

arc 2 **WRD V**

if <GETR last> = 5 then false
 else if <GETR last> = 1
 then SETR sum <GETR sum - 1>
else SETR sum <GETR sum + GETR last>
SETR last 5
TO NextCharacter

arc 3 **WRD X**

if <GETR last> = 1
 then {SETR sum <GETR sum - 1>
 SETR last 10
 SETR count 3 }
 count *wird auf 3 gesetzt damit nicht noch ein X auftreten kann*
else if <GETR last> = 5 then false
else if <GETR last> = 10 & <GETR count> = 3 then false
else if <GETR last> = 10
 then {SETR sum <GETR sum + 10>
 <SETR count <GETR count + 1>}
else {SETR sum <GETR sum + GETR last>
 SETR last 10
 SETR count 1}
TO NextCharacter

arc 4 **WRD L**

if <GETR last> = 5 | 50 then false
 else if <GETR last> = 1 | 10
 then {SETR sum <GETR sum - GETR last>
else SETR sum <GETR sum + GETR last>
SETR last 50
TO NextCharacter

arc 5 **WRD C**

```
if <GETR last> = 1 | 10
        then {SETR sum  <GETR sum - GETR last>
        SETR last 100
        SETR count 3 }
else if   <GETR last> = 5 | 50 then false
else if <GETR last> = 100 & <GETR count> =3 then false
else if <GETR last> = 100
        then {SETR sum  <GETR sum + 100>
        <SETR count <GETR count + 1>}
else {SETR sum  <GETR sum + GETR last>
        SETR last  100
        SETR count 1}
TO  NextCharacter
```

arc 6 **WRD D**

```
if <GETR last> = 5 | 50 | 500 then false
 else if <GETR last> = 1 | 10 | 100
        then {SETR sum  <GETR sum - GETR last>
else  SETR sum  <GETR sum + GETR last>
SETR last  50
TO  NextCharacter
```

arc 7 **WRD M**

```
if <GETR last> = 1 | 10 | 100
        then {SETR sum  <GETR sum - GETR last>
        SETR last 1000
        SETR count 3 }
```

```
else if   <GETR last> = 5 | 50 | 500 then false
else if <GETR last> = 1000 & <GETR count> =3 then false
else if <GETR last> = 1000
        then {SETR sum  <GETR sum + 1000>
        <SETR count <GETR count + 1>}
else {SETR sum  <GETR sum + GETR last>
        SETR last  1000
        SETR count 1}
TO  NextCharacter
```

arc 8 **WRD Leerzeichen**

```
Ende der Zahl erreicht
SETR sum  <GETR sum + GETR last>
SETR  result <GETR sum>
TO End
```

arc 9 **JUMP**

```
SETR result  "Fehler"
TO End
```

node **End**

arc **POP**

```
BUILDQ + result
```

2. Trace für die Zahl MCMLXIV

P:	Word:	Configurations:
1	**M**	*node* **RomanNumber** *arc:* JUMP *action* SETR last 0 SETR count 0 SETR sum 0 *transition* TO NextCharacter *node* **NextCharacter** *arc 1:* WRD I fails *arc 2:* WRD V fails *arc 3:* WRD X fails *arc 4:* WRD L fails *arc 5:* WRD C fails *arc 6:* WRD D fails *arc 7:* WRD M *test* *action* SETR sum <0 + 0> SETR last 1000 SETR count 1 *transition* TO NextCharacter
2	**C**	*node* **NextCharacter** *arc 1:* WRD I fails *arc 2:* WRD V fails *arc 3:* WRD X fails *arc 4:* WRD L fails

```
        arc 5: WRD C
        test
        action SETR sum = <0 + 1000>
        SETR last 100
        SETR count 1
        transition TO NextCharacter

3   M   node NextCharacter
        arc 1: WRD I fails
        arc 2: WRD V fails
        arc 3: WRD X fails
        arc 4: WRD L fails
        arc 5: WRD C fails
        arc 6: WRD D fails
        arc 7: WRD M
        test <GETR last> = 100
        action SETR sum  <1000 - 100>
        SETR last 1000
        SETR count 1
        transition TO NextCharacter

4   L   node NextCharacter
        arc 1: WRD I fails
        arc 2: WRD V fails
        arc 3: WRD X fails
        arc 4: WRD L
        test
        action SETR sum <900 + 1000>
        SETR last 50
        transition TO NextCharacter
```

```
5     X     node NextCharacter
              arc 1: WRD I fails
              arc 2: WRD V fails
              arc 3: WRD X
              test
              action SETR sum  <1900 + 50>
              SETR last 10
              SETR count 1
              transition TO NextCharacter

6     I     node NextCharacter
              arc 1: WRD I
              test
              action SETR sum  <1950 + 10>
              SETR last 1
              SETR count 1
              transition TO NextCharacter

7     V     node NextCharacter
              arc 1: WRD I fails
              arc 2: WRD V
              test <GETR last> = 1
              action SETR sum <1960 - 1>
              SETR last 5
              transition TO NextCharacter

      leer  node NextCharacter
              arc leer
              action SETR sum <1959 + 5>
              SETR result 1964
              transition TO End
```

Lösung 12: Slot-Filler Parser für Dependenzgrammatiken (PT-9)

A:	Segment:	Left margin:	Right margin:	Head bulletin:	Filler bulletin:	Discontinuous slot?
(1)	**wird**	1	1	-	-	N

(lexem[passiv'] kategorie[verb] verbtyp[passiv] form[finit] stellungstyp[verb_front, verb_zweit]
(> slot[oblig] rolle[praed_nukleus] kategorie[verb] verbtyp[vollverb] form[partizip]))

(2)	**wird**	1	1	-	-	N

(lexem[futur'] kategorie[verb] verbtyp[futur] form[finit] stellungstyp[verb_front, verb_zweit]
(> slot[oblig] rolle[praed_nukleus] kategorie[verb] verbtyp[vollverb, passiv,modal,perfekt]
form[infinitiv]))

(3)	**analysiert**	2	2	-	-	N

(lexem[analysieren] kategorie[verb] verbtyp[vollverb] form[partizip] perfekt[haben])

(4)	**wird analysiert**	1	2	1	3	N

(lexem[passiv'] kategorie[verb] verbtyp[passiv] form[finit] stellungstyp[verb_front, verb_zweit]
(> rolle[praed_nukleus] lexem[analysieren] kategorie[verb] verbtyp[vollverb] form[partizip]))

(5)	**worden**	3	3	-	-	N

(lexem[passiv'] kategorie[verb] verbtyp[passiv] form[partizip] perfekt[sein] stellungstyp[verb_end]
(< slot[nukleus] rolle[praed_nukleus] kategorie[verb] verbtyp[vollverb] form[partizip]
angrenzend[+]))

(6)	**analysiert worden**	2	3	5	3-	N

(lexem[passiv'] kategorie[verb] verbtyp[passiv] form[partizip] perfekt[sein] stellungstyp[verb_end]
(<] rolle[praed_nukleus] lexem[analysieren] kategorie[verb] verbtyp[vollverb] form[partizip]
angrenzend[+]))

(7)	**sein**	4	4	-	-	N

(lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[infinitiv] perfekt[sein] stellungstyp[verb_end]
(< slot[oblig] rolle[praed_nukleus] kategorie[verb] verbtyp[vollverb, passiv] form[partizip]
perfekt[C] angrenzend[+]))

(8)	**analysiert worden sein**	2	4	7	6	Y

(lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[infinitiv] perfekt[sein] stellungstyp[verb_end]
(< rolle[praed_nukleus] lexem[passiv'] kategorie[verb] verbtyp[passiv] form[partizip]
perfekt[sein] angrenzend[+]
(<] rolle[praed_nukleus] lexem[analysieren] kategorie[verb] verbtyp[vollverb] form[partizip]
angrenzend[+])))

(9)	**wird analysiert worden sein**	1	4	2	8	Y

(lexem[futur'] kategorie[verb] verbtyp[futur] form[finit] stellungstyp[verb_front, verb_zweit]
(> rolle[praed_nukleus] lexem[perfekt'] kategorie[verb] verbtyp[perfekt] form[infinitiv]
perfekt[sein] stellungstyp[verb_end]
(< rolle[praed_nukleus] lexem[passiv'] kategorie[verb] verbtyp[passiv] form[partizip]
perfekt[sein] angrenzend[+]
(<] rolle[praed_nukleus] lexem[analysieren] kategorie[verb] verbtyp[vollverb]
form[partizip] angrenzend[+]))))