Peter Hellwig: Kurs Maschinelle Syntaxanalyse (Parsing)

Aufgaben und Lösungen 1

Diese Unterrichtsmaterialien sind frei verwendbar.

Aufgabe 1: Die Idee der generativen Grammatik

1. Schreiben Sie eine Phrasenstrukturgrammatik mit Regeln und Lexikon, die folgendes Fragment des Englischen abdeckt, einschließlich der in Satz (1) und (3) enthaltenen Mehrdeutigkeit, die aber in Satz (2) und (4) nicht auftritt.

(1) they love visiting relatives
(2) they visit annoying relatives
(3) students hate annoying professors.
(4) students hate annoying their professors

2. Zeigen Sie welche Phrasenstrukturbäume für die Sätze (1) und (2) von Ihrer Grammatik generiert werden.

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

Gegeben sei das Fragment der englischen Phrasenstrukturgrammatik, welches Sie in Aufgabe 1 geschrieben haben. Bitte verwenden Sie die Form, die in Lösung 1 steht.

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

"students hate annoying their professors"

2. Kann man den Parser effizienter machen, indem man die Grammatik ändert, und wie?

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

1. Entnehmen Sie die Grammatik des Englischfragments 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

   "the students hate annoying professors"

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)

Um Vergleiche zu ermöglichen, sei noch einmal derselbe Satz wie in Aufgabe 2 zu parsen, wobei wieder die Grammatik aus der Lösung 1 verwendet werden soll:

" students hate annoying their professors"

1. Folgen Sie der Beschreibung von PT-2 im Skript und zeigen Sie, wie der Satz im WORKING SPACE abgearbeitet wird.
2. Vergleichen Sie das Ergebnis mit der Lösung der Aufgabe 2. Welche Vor- und Nachteile von PT-1 und PT-2 lassen sich daran ablesen?

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

Gegeben sei zum letzten Mal (versprochen!) die Grammatik aus Lösung 2:

(R-1)	**S → NP + VP**	Det	= {*the, their*}
(R-2)	**NP → Noun**	Adj	= {*loving, hating, annoying, visiting*}
(R-3)	**NP → Det + Noun**	Noun	= {*relatives, students, professors*}
(R-4)	**NP → Adj + Noun**	Pron	= {*they*}
(R-5)	**NP → Pron**	Vt	= {*love, hate, annoy, visit*}
(R-6)	**VP → Vt + NP**	Vtger	= {*love, hate*}
(R-7)	**VP → Vtger + GP**	Ger	= {*loving, hating, annoying, visiting*}
(R-8)	**GP → Ger + NP**		

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:

   " students hate annoying their professors "
3. Vergleichen Sie PT-1 (Aufgabe 2), PT-2 (Aufgabe 4) und PT-3. Ist PT-3 effizienter und um wie viel?

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

Gegeben sei folgende links-rekursive Grammatik mit Lexikon:

Regeln:

(R-1)	SATZ -> NOGR VERB
(R-2)	NOGR -> ADJE NOGR
(R-3)	NOGR -> NOGR RELS
(R-4)	RELS -> RELW VERB

Lexikon:

Ideen	NOGR
begeistern	VERB
fehlen	VERB
farblose	ADJE
neue	ADJE
die	RELW

Erstellen Sie Input table und Working Table von PT-4, wie im Skript beschrieben, für den Satz

"neue Ideen die begeistern fehlen"

Es gibt zwei Ergebnisse. Woran liegt das? Tip: Suchen Sie alle Einträge mit fertigen Produktionen heraus (d.s. solche mit Punkt am Ende). Machen Sie sich graphisch klar, von wo bis wo jede dieser Produktionen reicht, und welche Segmente der Completor jeweils zu einem größeren Segment zusammengefasst hat. Ist es linguistisch gerechtfertigt, dass es zwei Lösungen gibt?

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

1. Hier ist eine Grammatik der deutschen Kardinalzahlen in Chomsky-Normalform. Fettgedruckte Kategorien markieren jeweils die dominierende Konstituente (den *Head*) in der Regel. Tiefgestellte Zahlen sind Subklassifizierungen. Kommata trennen alternative Werte voneinander ab. In der Working Table werden diese Disjunktionen in Einzelkategorien aufgelöst.

Lexikon:	**Regeln:**	**Bereich der Regeln:**
Z_1 = {ein, zwei, drei, vier, fünf, sechs, sieben, acht, neun}	(R-1) Z_2 -> Z_3 + **zehn**	11 - 19
Z_2 = {zehn, elf, zwölf}	(R-2) Z_7 -> Z_4 + **zig**	20 - 90, außer 30
Z_3 = {drei, vier, fünf, sech, sieb, acht, neun}	(R-3) Z_7 -> drei + **ssig**	30
Z_4 = {zwan, vier, fünf, sech, sieb, acht, neun}	(R-4) Z_8 -> U + $\mathbf{Z_7}$	21 - 99
Z_5 = {hundert}	(R-5) Z_9 -> Z_1 + $\mathbf{Z_5}$	100 - 900
Z_6 = {tausend}	(R-6) Z_{10} -> Z_2 + $\mathbf{Z_5}$	1100 - 1900 (elfhundert)
drei = {drei}	(R-7) Z_{11} -> $\mathbf{Z_{5,9}}$ + $Z_{1,2,7,8}$	100 - 199, 200 - 299 usw. bis 999
zehn = {zehn}	(R-8) Z_{12} -> $\mathbf{Z_{10}}$ + $Z_{1,2,7,8}$	1101 - 1999 (elfhundertein)
zig = {zig}	(R-9) Z_{13} -> $Z_{1,2,7,8,9,11}$ + $\mathbf{Z_6}$	1000 - 999000
ssig = {ssig}	(R-10) Z_{14} -> $\mathbf{Z_{6,13}}$ + $Z_{1,2,7,8,9,11}$	1999 - 999999
und = {und}	(R-11) U -> Z_1 + **und**	1 - 9 und ... zig

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

zwei-hundert-zwei-und-zwan-zig-tausend-vier-hundert-sieb-zehn

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

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

Nehmen Sie noch einmal die Grammatik von Aufgabe 6:

Regeln:

(R-1)	SATZ	-> NOGR VERB
(R-2)	NOGR	-> ADJE NOGR
(R-3)	NOGR	-> NOGR RELS
(R-4)	RELS	-> RELW VERB

Lexikon:

Ideen	NOGR
begeistern	VERB
fehlen	VERB
farblose	ADJE
neue	ADJE
die	RELW

1. Schreiben Sie eine Kategoriegrammatik, die eine NOGR mit beliebig vielen Adjektiven aber nur einem Relativsatz akzeptiert und ansonsten stark äquivalent zu der obigen Grammatik ist. Dazu müssen Sie zuerst versuchen, aus den Regeln 1 - 4 lexikalische Kategorien abzuleiten. Ob dabei die linke oder die rechte unmittelbare Konstituente komplex wird, während die andere einfach bleibt, müssen Sie ausprobieren. Schließlich weisen Sie den Wörtern im Lexikon die neuen Kategorien zu. Fertig ist die Kategorialgrammatik.

2. Zeigen Sie anhand der Working table wie PT-5 (ohne Prädiktor, wie in Stunde 8 beschrieben), den folgenden Satz parst:

   "neue Ideen die begeistern fehlen"

3. Vergleichen Sie die Performanz von PT-4 in Lösung 6 mit ihrer Lösung. Welcher Parser ist effizienter und warum?

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

Gegeben die Grammatik aus Lösung 1:

Regeln:

(R-1)	**S → NP + VP**
(R-2)	**NP → Noun**
(R-3)	**NP → Det + Noun**
(R-4)	**NP → Adj + Noun**
(R-5)	**NP → Pron**
(R-6)	**VP → Vt + NP**
(R-7)	**VP → Vtger + GP**
(R-8)	**GP → Ger + NP**

Lexikon:

Det	**= {*the, their*}**
Adj	**= {*loving, hating, annoying, visiting*}**
Noun	**= {*relatives, students, professors*}**
Pron	**= {*they*}**
Vt	**= {*love, hate, annoy, visit*}**
Vtger	**= {*love, hate*}**
Ger	**= {*loving, hating, annoying, visiting*}**

1. Überführen Sie nach Anleitung des Skripts diese Grammatik in eine Aktionstabelle und eine Sprungtabelle für den tabellen-gesteuerten Shift-Reduce Parser.

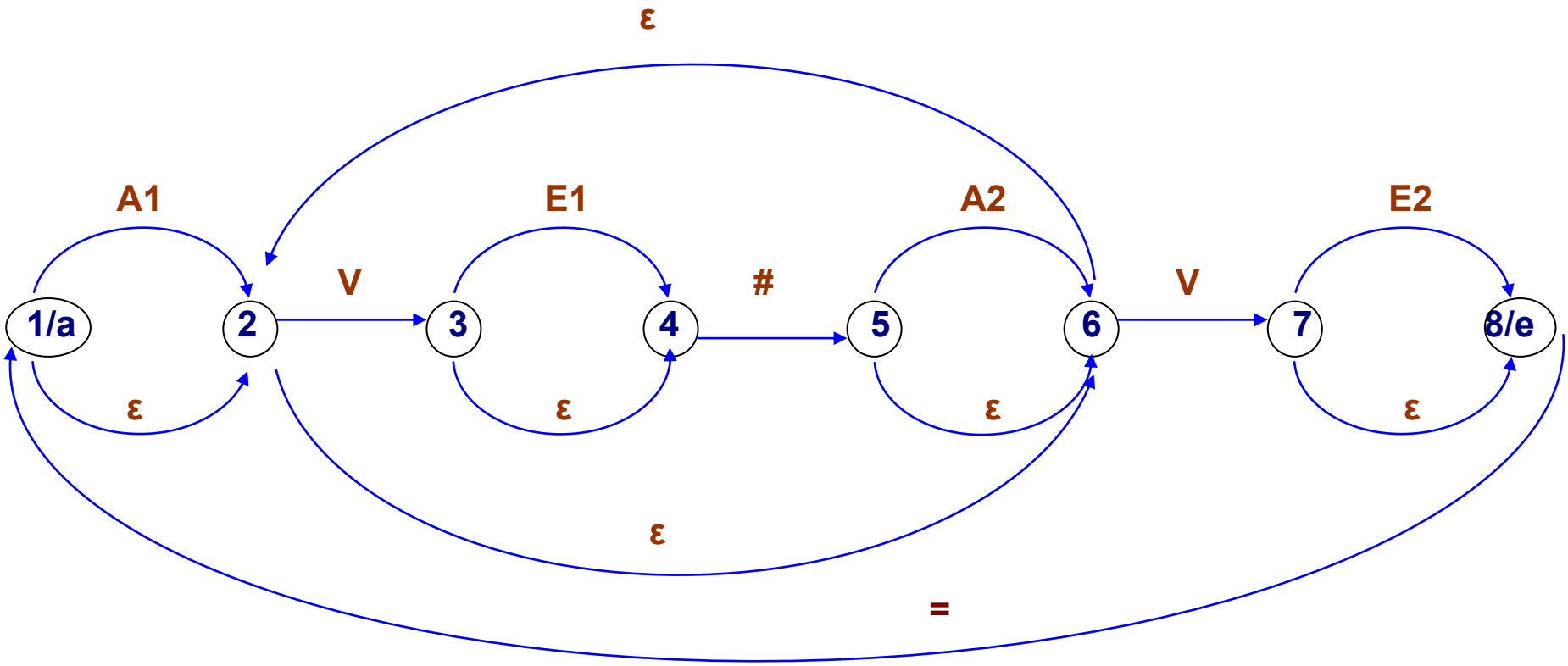
2. Prüfen Sie Ihre Tabellen, indem Sie den folgenden Satz verarbeiten:

" students hate annoying their professors"

3. Vergleichen Sie das Ergebnis mit dem PT-1 (Lösung 2). Was beobachten Sie?

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

Folgendes nicht-deterministische Übergangsnetzwerk stellt die Bildung von Wörtern aus Silben dar. Silben bestehen aus konsonantischem Anlaut (A), vokalischem Silbengipfel (V) und konsonantischem Auslaut (E). dabei macht es einen Unterschied, ob ein Silbenanlaut am Anfang des Wortes (A1) oder in der Mitte des Wortes (A2) steht und ob ein Silbenauslaut vor einer anderen Silbe (E1) oder am Ende des Wortes (E2) steht.



'/a' Eingang, ' /e' Ausgang, ' #' bedeutet Silbengrenze, '=' bedeutet Wortgrenze (auch in Komposita)

Das Netz entspricht folgenden regulären Ausdrücken:

({A1? (V E1? # A2?)* V E2?} =) *

A1 = **b | bl | br | c | ch | chl | chr | cl | cr | d | dr | dsch | f | fl | fr | g | gh | gl | gn | gr | h | j | k | kl | kn | kr | l | m | n | p | pf | pfl | pfr | ph | phr | pl | pr | ps | qu | r | rh | s | sch | schl | schm | schn | schr | schw | sk | skl | skr | sl | sm | sn | sp | sph | spl | spr | st | str | sz | t | th | thr | tr | tsch | v | w | wr | x | z | zw**

A2 = **b | c | ch | d | f | g | h | j | k | l | m | n | p | qu | r | s | sch | ß | st | t | v | w | x | z**

V = **a | ä | aa | ae | ah | äh | ai | äo | au | äu | ay | e | ea | ee | eh | ei | eih | eo | eu | ey | i | ia | ie | ieh | ih | io | o | ö | oa | oh | öh | oi | oo | öo | ou | ou | oua | oui | oy | u | ü | ua | uh | üh | ui | y**

E1 = **ch | c-k | d | f | g | k | l | m | n | p | ph | r | s | sch | ß | st | t | th | v | x | z**

E2 = **b | bs | bsch | bst | bt | ch | chs | chst | cht | chts | chz | chzt | ck | cks | ckst | ckt | d | ds | dst | dt | f | ff | ffs | ffst | fft | fs | fst | ft | fts | fzt | g | gd | gg | ggst | ggt | gs | gst | gt | k | ks | kst | kt | kts | l | lb | lbs | lbst | lbt | lch | ld | lds | lf | lfs | lfst | lft | lg | lgs | lgst | lgt | lk | lks | lkst | lkt | ll | lls | llst | llt | lm | lms | ln | lnd | lp | ls | lsch | lscht | lst | lt | lts | ltst | lz | lzt | m | mb | mbst | mbt | md | mf | mm | mms | mmst | mmt | mp | mpf | mpfst | mpft | mps | mpst | ms | msch | mschst | mscht | mst | mt | mts | n | nch | nd | nds | ndt | nf | nft | nfts | ng | ngs | ngst | ngt | nk | nks | nkst | nkt | nkts | nn | nns | nnst | nnt | ns | nsch | nst | nt | nts | nz | nzt | p | pf | pfs | pfst | pft | ph | pp | pps | ppst | ppt | ps | pt | pts | r | rb | rbs | rbst | rbt | rch | rchst | rcht | rd | rds | rf | rfs | rfst | rft | rg | rgs | rgst | rgt | rk | rks | rkst | rkt | rl | rls | rlst | rlt | rm | rmst | rmt | rn | rnd | rns | rnst | rnt | rp | rps | rpt | rr | rrschst | rrscht | rrst | rrt | rs | rsch | rst | rt | rts | rts | rv | rvst | rvt | rz | rzt | s | sch | schst | scht | sk | ß | st | t | th | ts | tsch | tt | tts | ttst | ttst | tz | tzt | v | vs | x | xt | z | zt | zz |**

1. Formen Sie das Netz in einen endlichen Erkenner um (d.i. eine Übergangstabelle, in der die Ausgangs- und Zielzustände Mengen von Zuständen aus dem obigen Netz entsprechen). Benutzen Sie als Namen für die neuen Zustände die durch '/ ' getrennten Namen der alten Zustände.

2. Zeigen Sie nacheinander, welche Übergänge Ihr Erkenner vollzieht, um folgenden Eingaben zu akzeptieren oder zurückzuweisen:

   wal#ten#de

   wald=en#de

   wal#te#nde

3. Reicht der endliche Erkenner zur Lösung der Aufgabe, Wörter in Silben zu parsen?

.

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

Lexikon:

```
ein      wrd[ein] cat[zahl] typ[2] zif[1]
eins     wrd[eins] cat[zahl] typ[1] zif[1]
zwei     wrd[zwei] cat[zahl] typ[1,2] zif[2]
drei     wrd[drei] cat[zahl] typ[1,2,3,4] zif[3]
vier     wrd[vier] cat[zahl] typ[1,2,3,4] zif[4]
fünf     wrd[fünf] cat[zahl] typ[1,2,3,4] zif[5]
sechs    wrd[sechs] cat[zahl] typ[1,2] zif[6]
sech     wrd[sech] cat[zahl] typ[3,4] zif[6]
sieben   wrd[sieben] cat[zahl] typ[1,2] zif[7]
sieb     wrd[sieb] cat[zahl] typ[3,4] zif[7]
acht     wrd[acht] cat[zahl] typ[1,2,3,4] zif[8]
neun     wrd[neun] cat[zahl] typ[1,2,3,4] zif[9]
zehn     wrd[zehn] cat[zahl] typ[1] zif[10]
elf      wrd[elf] cat[zahl] typ[1] zif[11]
zwölf    wrd[zwölf] cat[zahl] typ[1] zif[12]
zwan     wrd[zwan] cat[zahl] typ[4] zif[2]
zig      wrd[zig] cat[zahl] typ[1] zif[0]
ssig     wrd[ssig] cat[zahl] typ[1] zif[0]
hundert  wrd[hundert] cat[zahl] typ[1] zif[100]
```

Aufgabe:

1. Schreiben Sie nach dem Vorbild von PT-8 im Skript ein ATN-Programm, das die deutschen Kardinalzahlen von "eins" bis "neunhundertneunundneunzig**"** akzeptiert und als Ziffern 1 bis 999 ausgibt. Halten Sie sich an den im Skript definierten Formalismus. Verwenden Sie das obige Lexikon. Dabei entspricht das Merkmal "wrd" dem Wert in einer WRD-Kante, "cat" dem Wert in einer CAT-Kante. Das Merkmal "typ" kann in Registern Verwendung finden. Die Werte bedeuten:

    1 - allein möglich

    2 - vor "hundert" und vor "und"

    3 - vor "zehn"

    4 - vor "zig" oder "ssig"

Das Merkmal "zif" kann in Registern Verwendung finden und enthält Ziffern.

2. Zeigen Sie, dass Ihr ATN-Parser für folgende Zahl funktioniert:

    vierhundertfünfunddreissig

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

Die folgende Dependenzgrammatik deckt deutsche Zahlen von *eins* bis *neunhundertneunundneunzig-tausendneunhundertneunundneunzig* ab. Bei den Template stehen zum Vergleich die entsprechenden Regeln der PSG in Chomsky-Normalform aus Aufgabe 7.

Morpho-syntaktisches Lexikon:

acht	(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_zig] wert[8])
drei	(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_ssig] wert[3])
ein	(kategorie[zahl] typ[ein, vor_und] wert[1])
elf	(kategorie[zahl] typ[zehn] wert[11])
fünf	(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_zig] wert[5])
hundert	(kategorie[zahl] typ[hundert] wert[100])
neun	(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_zig] wert[9])
sech	(kategorie[zahl] typ[vor_zehn, vor_zig] wert[6])
sechs	(kategorie[zahl] typ[ein, vor_und] wert[6])
sieb	(kategorie[zahl] typ[vor_zehn, vor_zig] wert[7])
sieben	(kategorie[zahl] typ[ein, vor_und] wert[7])
ssig	(kategorie[zahl] typ[ssig] wert[10])
tausend	(kategorie[zahl] typ[tausend] wert[1000])
vier	(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_zig] wert[4])
zehn	(kategorie[zahl] typ[zehn] wert[10])
zig	(kategorie[zahl] typ[zig] wert[10])
zwan	(kategorie[zahl] typ[vor_zig] wert[2])
zwei	(kategorie[zahl] typ[ein, vor_und] wert[2])
zwölf	(kategorie[zahl] typ[zehn] wert[12])
und	(kategorie[zahl] typ[und])

Templates:	**Regeln Aufg. 7:**	**Beispiel:**
(template[+addit_zehn] kategorie[zahl] typ[zehn] (< SLOT rolle[ADDIT] kategorie[zahl] typ[vor_zehn]));	R-1	SIEBzehn
(template[+multp_zig] kategorie[zahl] typ[zig] (< SLOT rolle[MULTP] kategorie[zahl] typ[vor_zig]));	R-2	SIEBzig
(template[+multp_ssig] kategorie[zahl] typ[ssig] (< SLOT rolle[MULTP] kategorie[zahl] typ[vor_ssig]));	R-3	DREIssig
(template[+addit_zig] kategorie[zahl] typ[zig,ssig] (< SLOT rolle[ADDIT] kategorie[zahl] typ[und]));	R-4	VIERUND(vier)zig
(template[+multp_hundert] kategorie[zahl] typ[hundert] (< SLOT rolle[MULTP] kategorie[zahl] typ[ein, zehn]));	R-5, R-6	VIERhundert; SIEBZEHNhundert
(template[+addit_hundert] kategorie[zahl] typ[hundert] (> SLOT rolle[ADDIT] kategorie[zahl] typ[ein, zehn, zig, ssig]));	R-7, R-8	hundertVIER; hundertUNDVIER; (elf)hundertSIEBZEHN
(template[+multp_tausend] kategorie[zahl] typ[tausend] (< SLOT rolle[MULTP] kategorie[zahl] typ[ein, zehn, zig, ssig, hundert]));	R-9	NEUNtausend NEUNZIGtausend
(template[+addit_tausend] kategorie[zahl] typ[tausend] (> SLOT rolle[ADDIT] kategorie[zahl] typ[ein,zehn.zug.ssig.hundert));	R-10	tausendDREI
(template[+und_z] kategorie[zahl] typ[und] (< SLOT kategorie[zahl] typ[vor_und]));	R-11	ZWEIund(zwanzig)

Synframes:

```
(kategorie[zahl] wert[10] typ[zehn]
    (complement[+addit_zehn]))

(kategorie[zahl] wert[10] typ[zig]
    (complement[+multp_zig])

(kategorie[zahl] wert[10] typ[ssig]
    (complement[+multp_ssig])

(kategorie[zahl] wert[10] typ[zig,ssig]
    (complement[+addit_zig])

(kategorie[zahl] wert[100] typ[hundert]
    (complement[+multp_hundert, +addit_hundert)

(kategorie[zahl] wert[1000] typ[tausend]
    (complement[+multp_tausend, +addit_tausend])

(kategorie[zahl] typ[und]
    (complement[+und_z])
```

Aufgabe:

1. Zeigen Sie nach dem Vorbild des Skripts, wie folgende Zahl von PT-9 analysiert wird:

vier-hundert-fünf-und-drei-ssig

2. Wie könnte man aus dem Ergebnisbaum die Zahl 435 errechnen?

Aufgabe 13: Parsing mit statistischen Methoden

Hier ist Phantasie gefragt: Erweitern Sie einen der prototypischen Parser PT-1 bis PT-9 um eine statistische Komponente. Zeigen Sie an einem Beispiel, wie sich der Parser ohne und mit Ihrer Erweiterung verhält. Besprechen Sie das Ergebnis.

- Es ist nicht verboten, zur Lösung dieser Aufgabe in Büchern oder im Internet zu recherchieren!!!

Aufgabe 14: 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

1. Die folgende Phrasenstrukturgrammatik erfüllt die Aufgabe:

Regeln:

(R-1) **S → NP + VP**

(R-2) **NP → Noun**

(R-3) **NP → Det + Noun**

(R-4) **NP → Adj + Noun**

(R-5) **NP → Pron**

(R-6) **VP → Vt + NP**

(R-7) **VP → Vtger + GP**

(R-8) **GP → Ger + NP**

Lexikon:

Det = {*the, their*}

Adj = {*loving, hating, annoying, visiting*}

Noun = {*relatives, students, professors*}

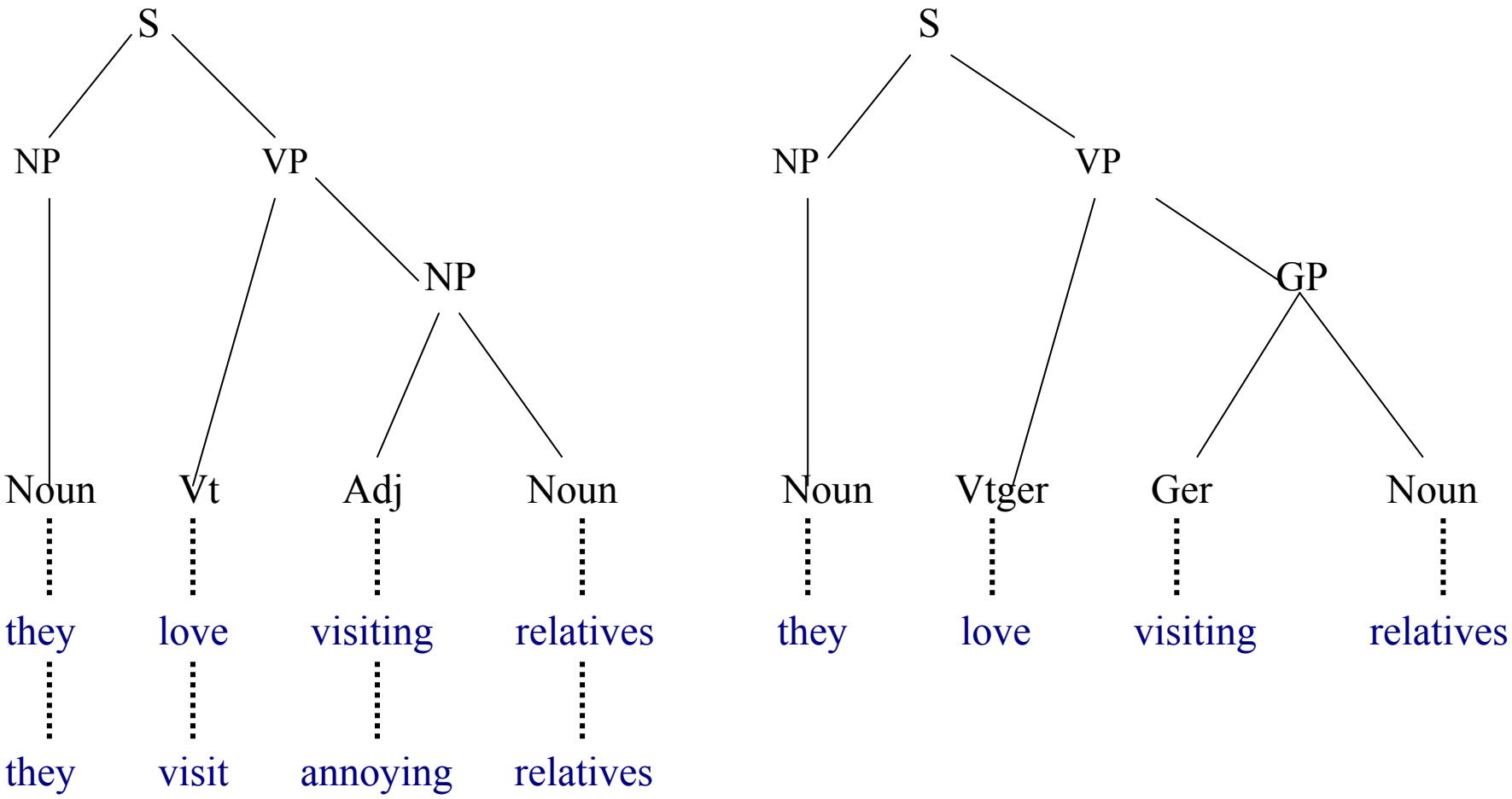
Pron = {*they*}

Vt = {*love, hate, annoy, visit*}

Vtger = {*love, hate*}

Ger = {*loving, hating, annoying, visiting*}

2. Phrasenstrukturbäume für den mehrdeutigen Satz (1) und den eindeutigen Satz (2):



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

INPUT TABLE nach der Lexikonphase:

Wörter:	students	hate	annoying	their	professors
terminale Kategorien:	Noun	Vt/Vtger	Adj/Ger	Det	Noun
Position:	1	2	3	4	5

WORKING SPACE **BACK STORE**

A	Derivation	Explanation	P	B	A	P	R
1	S	State at start	1	0			
2	NP + VP	expansion R-1	1				
3	Noun + VP	expansion R-2	1	1	2	1	3
4	VP	recognized Noun	2				
5	Vt + NP	expansion R-6	2	2	4	2	7
6	NP	recognized Vt	3				
7	Noun	expansion R-2	3	3	6	3	3
6	NP	back to A=6 P=3 R=3	3	2			
7	Det + Noun	expansion R-3	3	3	6	3	4
6	NP	back to A=6 P=3 R=4	3	2			
7	Adj + Noun	expansion R-4	3	3	6	3	5
8	Noun	recognized Adj	4				

A	Derivation	Explanation	P	B	A	P	R
6	NP	back to A=6 P=3 R=5	3	2			
7	Pron	expansion R-5	3				
4	VP	back to A=4 P=2 R=7	2	1			
5	Vtger + GP	expansion R-7	2				
6	GP	recognized Vtger	3				
7	Ger + NP	expansion R-8	3				
8	NP	recognized Ger	4				
9	Noun	expansion R-2	4	2	8	4	3
8	NP	back to A=8 P=4 R=3	4	1			
9	Det + Noun	expansion R-3	4	2	8	4	4
10	Noun	recognized Det	5				
11	-	recognized Noun	6				

Lösung gefunden. Parse (Regelanwendungen): 1, 2, 7, 8, 3 - Es geht aber weiter!

8	NP	back to A=8, P=4, R=4	4	1			
9	Adj + Noun	expansion R-4	4	2	8	4	5
8	NP	back to A=8, P=4, R=5	4	1			
9	Pron	expansion R-5	4				
2	NP + VP	back to A=2, P=1, R=3	1	0			
3	Det + N + VP	expansion R-3	1	1	2	1	4
2	NP + VP	back to A=2, P=1, R=4	1	0			
3	Adj + N + VP	expansion R - 4	1	1	2	1	5
2	NP + VP	back to A=2, P=1, R=5	1	0			
3	Pron + VP		1	0			

2. Wie kann man den Parser effizienter machen, indem man die Grammatik ändert?

Man müsste erreichen, dass die Regeln möglichst konkret auf präterminale Kategorien bezogen sind. Statt

S → NP + VP

und

VP → Vt + NP
VP → Vtger + GP

könnte man gleich schreiben

S → NP + Vt + NP
S → NP + Vtger + GP

Aber viel bringt das nicht. Eine prinzipielle Möglichkeit werden wir mit der Umformung der Grammatik in Greibach-Normalform (PT-3) kennenlernen.

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

Trace des Parsers PT-1 mit Stack durch rekursiven Programmaufruf

parser()

Sentence = the students hate annoying professors
Preterminals = Det+Noun+Vt/Vtger+Adj/Ger+Noun
Constituents=S
Position=1
Parse=0

expansion(S, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 1, 0)
/* recognize*/
/* assess result */
Success=0
/* expansion */
SaveConstituents=S
SaveParse=0
SavePosition=1
Wanted=S
Loop R=1
Constituents=NP+VP
Parse=1

expansion(NP+VP, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 1, 1)
/* recognize*/
/* assess result */
Success=0
/* expansion */
SaveConstituents=NP+ VP
SaveParse=1
SavePosition=1
Wanted=NP
Loop R=1 bis 2
Constituents=Noun+VP
Parse=1-2

expansion(Noun+VP, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 1, 1-2)
/* recognize*/
/* assess result */
Success=0
/* expansion */
SaveConstituents=Noun+VP
SaveParse=1-2
SavePosition=1
Wanted=Noun
Loop R = 1 bis Regelende
return (Success=0)
end expansion

	/* **backtracking** */ Constituents=NP+VP SaveParse=1 SavePosition=1 Loop R=3 Constituents=Det+Noun+VP Parse=1-3	
		expansion(Det+Noun+VP, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 1, 1-3) /* **recognize** */ Loop Constituents=Noun+VP Position=2 Constituents=VP Position=3 Ende Loop /* **assess result** */ Success=0 /* **expansion** */ SaveConstituents=VP SaveParse=1-3 SavePosition=3 Wanted=VP Loop R=1 bis 6 Constituents=Vt+NP Parse=1-3-6

			expansion(Vt+NP, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 3, 1-3-6) **/* recognize */** Loop Constituents=NP Position=4 Ende Loop **/* assess result */** Success=0 **/* expansion */** SaveConstituents=NP SaveParse=1-3-6 SavePosition=4 Wanted=NP Loop R=1 bis 2 Constituents=Noun Parse=1-3-6-2	
				expansion(Noun, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 4, 1-3-6-2) **/* recognize*/** **/* assess result */** Success=0 **/* expansion */** SaveConstituents=Noun SaveParse=1-3-6-2 SavePosition=4 Wanted=Noun Loop R = 1 bis Regelende return (Success=0) end expansion

			/* **backtracking** */ Constituents=NP SaveParse=1-3-6 SavePosition=4 Loop R=3 Constituents=Det+Noun Parse=1-3-6-3	
				expansion(Det+Noun, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 4, 1-3-6-2) /* **recognize***/ /* **assess result** */ Success=0 /* **expansion** */ SaveConstituents=Det SaveParse=1-3-6-2 SavePosition=4 Wanted=Det Loop R = 1 bis Regelende return (Success=0) end expansion
			/* **backtracking** */ Constituents=NP SaveParse=1-3-6 SavePosition=4 Loop R=4 Constituents=Adj+Noun Parse=1-3-6-4	

				expansion(Adj+Noun, Det+Noun+Vt/Vtger+Adj/Ger+Noun, 4, 1-3-6-4) **/* recognize */** Loop Constituents=Noun Position=5 Constituents= Position=6 Ende Loop **/* assess result */** print_parse_tree(1-3-6-4) return (Success=1) end expansion
			Success=1, break return (Success=1) end expansion	
		Success=1, break return (Success=1) end expansion		
	Success=1, break return (Success=1) end expansion			
Success=1, break return (Success=1) end expansion				
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: " students hate annoying their professors "

WORKING AREA

Pos.	Input	Lexicon	Derivation	Applied rules ("parse")
1	students	Noun	**S** **NP + VP** **Noun + VP** **Det + Noun + VP** **Adj + Noun + VP** **Pron + VP**	- 1 1,2 1,3 1,4 1,5
			VP	1,2
2	hate	Vt/Vtger	**Vt + NP** **Vtger + GP**	1,2,6 1,2,7
			NP **GP**	1,2,6 1,2,7

3	annoying	Adj/Ger	**Noun** **Det + Noun** **Adj + Noun** **Pron** **Ger + NP**	1,2,6,2 1,2,6,3 1,2,6,4 1,2,6,5 1,2,6,8
			Noun **NP**	1,2,6,4 1,2,6,8
4	their	Det	**Noun** **Noun** **Det + Noun** **Adj + Noun** **Pron**	1,2,6,4 1,2,6,8,2 1,2,6,8,3 1,2,6,8,4 1,2,6,8,5
			Noun	1,2,6,8,3
5	professors	Noun	**Noun**	1,2,6,8,3
			-	**1,2,6,8,3**

Der endgültige Parse lautet 1,2,6,8,3

Vergleich zwischen PT-1 (Aufgabe 2):

Beide Parser vollziehen 24 Schritte (Expansionen mit anschließendem Erkennen).

Sie sind also gleich aufwändig.

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

Grammatik aus Lösung 2:

(R-1)	**S → NP + VP**	Det	= {*the, their*}
(R-2)	**NP → Noun**	Adj	= {*loving, hating, annoying, visiting*}
(R-3)	**NP → Det + Noun**	Noun	= {*relatives, students, professors*}
(R-4)	**NP → Adj + Noun**	Pron	= {*they*}
(R-5)	**NP → Pron**	Vt	= {*love, hate, annoy, visit*}
(R-6)	**VP → Vt + NP**	Vtger	= {*love, hate*}
(R-7)	**VP → Vtger + GP**	Ger	= {*loving, hating, annoying, visiting*}
(R-8)	**GP → Ger + NP**		

Umformung der Grammatik in Greibach-Normalform:

(R-1)	**(S, Noun)**	|	**VP**
(R-2)	**(S, Det)**	|	**N + VP**
(R-3)	**(S, Adj)**	|	**N + VP**
(R-4)	**(S, Pron)**	|	**VP**
(R-5)	**(VP, Vt)**	|	**NP**
(R-6)	**(VP, Vtger)**	|	**GP**
(R-7)	**(N, Noun)**	|	**-**
(R-8)	**(NP, Noun)**	|	**-**
(R-9)	**(NP, Det)**	|	**N**
(R-10)	**(NP, Adj)**	|	**N**
(R-11)	**(NP, Pron)**	|	**-**
(R-12)	**(GP, Ger)**	|	**NP**

EINGABE: ″students hate annoying their professors ″

WORKING AREA

Pos.	Input	Derivation	Preterminal	Prediction	Applied rules ("parse")
1	**students**	**S**	**Noun**	**VP**	1
2	**hate**	**VP**	**Vt** **Vtger**	**NP** **GP**	1,5 1,6
3	**annoying**	**NP** **GP**	**Adj** **Ger**	**N** **NP**	1,5,10 1,6,12
4	**their**	**N** **NP**	**Det** **N**	*fail*	 1,6,12,9
5	**professors**	**N**	**Noun**	-	1,6,12,9,8

Vergleich PT-1, PT-2, PT-3:

PT-1 und PT-2 brauchten 24 Schritte (Expansionen, Erkennen)

Der Predictive Analyzer PT-4 braucht nur 8 Schritte; er ist also um 2/3 schneller.

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

Regeln:

(R-1) **SATZ -> NOGR VERB**

(R-2) **NOGR -> ADJE NOGR**

(R-3) **NOGR -> NOGR RELS**

(R-4) **RELS -> RELW VERB**

Lexikon:

Ideen	**NOGR**
begeistern	**VERB**
fehlen	**VERB**
farblose	**ADJE**
neue	**ADJE**
die	**RELW**

EINGABE	neue	Ideen	die	begeistern	fehlen
LEXIKON	**ADJE**	**NOGR**	**RELW**	**VERB**	**VERB**
POSITION	0 1	1 2	2 3	3 4	4 5

WORKING TABLE

Section 0:

	Divided productions	**Left mgn.**	**Right mgn.**	**Explanation**
(1)	**# -> .SATZ**	0	0	starting state

(2)	**SATZ -> .NOGR VERB**	0	0	predictor for (1) by R-1
(3)	**NOGR -> .ADJE NOGR**	0	0	predictor for (2) by R-2
(4)	**NOGR -> .NOGR RELS**	0	0	predictor for (2) by R-3
(*)	**NOGR -> .ADJE NOGR**	0	0	predictor for (4) by R-2
(*)	**NOGR -> .NOGR RELS**	0	0	predictor for (4) by R-3

Section 1:

(5)	**NOGR -> ADJE .NOGR**	0	1	scanner for(3), *neue*
(6)	**NOGR -> .ADJE NOGR**	1	1	predictor for (5) by R-2
(7)	**NOGR -> .NOGR RELS**	1	1	predictor for (5) by R-3
(*)	**NOGR -> .ADJE NOGR**	1	1	predictor for (7) by R-2
(*)	**NOGR -> .NOGR RELS**	1	1	predictor for (7) by R-3

Section 2:

(8)	**NOGR -> ADJE NOGR.**	0	2	scanner for (5), *Ideen*
(9)	**NOGR -> NOGR. RELS**	1	2	scanner for (7), *Ideen*
(10)	**SATZ -> NOGR .VERB**	0	2	completor (8) in (2)
(11)	**NOGR -> NOGR .RELS**	0	2	completor (8) in (4)
(12)	**RELS -> .RELW VERB**	2	2	predictor for (8) by R-4
(*)	**RELS -> .RELW VERB**	2	2	predictor for (11 by R-4

Section 3:

(13)	**RELS -> RELW. VERB**	2	3	scanner for (12), *die*

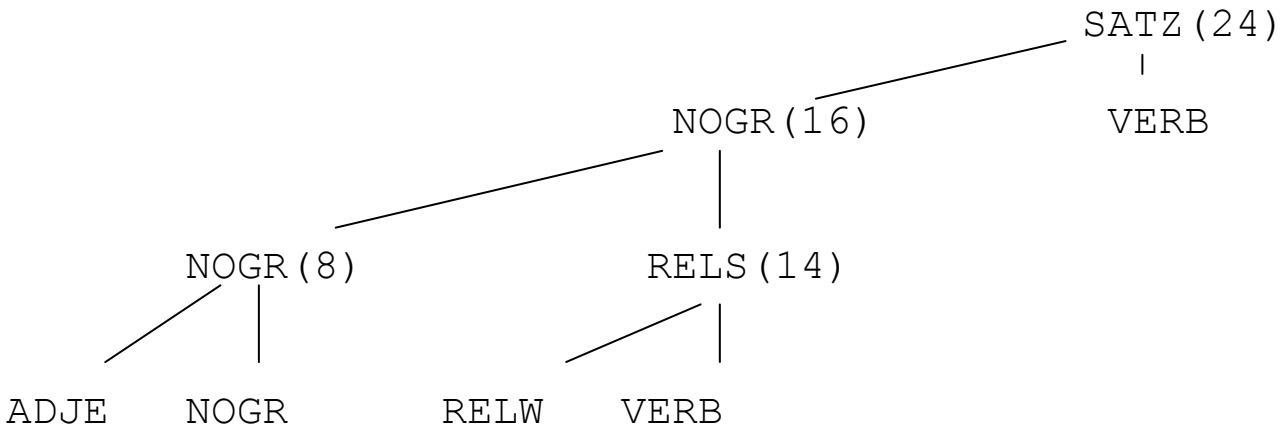
Section 4:

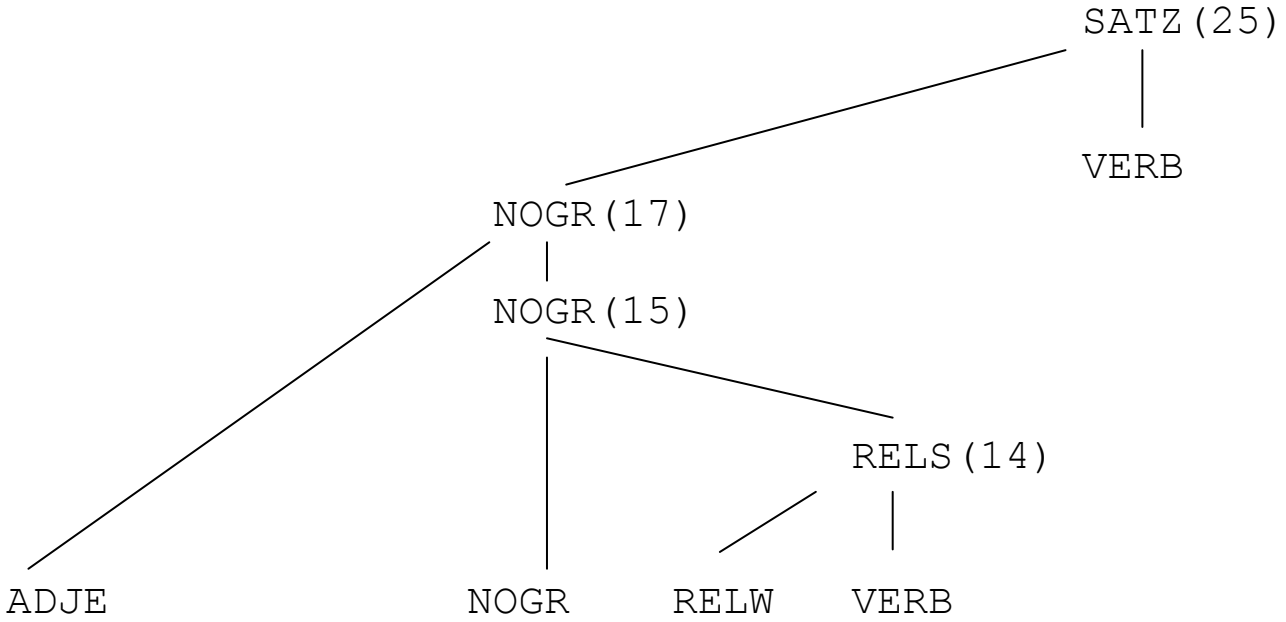
(14)	**RELS -> RELW VERB.**	2	4	scanner for (13), *begeistern*
(15)	**NOGR -> NOGR RELS.**	1	4	completor (14) in (9)
(16)	**NOGR -> NOGR RELS.**	0	4	completor (14) in (11)
(17)	**NOGR -> ADJE NOGR.**	0	4	completor (15) in (5)
(18)	**NOGR -> NOGR .RELS**	1	4	completor (15) in (7)
(19)	**SATZ -> NOGR .VERB**	0	4	completor (16) in (2)
(20)	**NOGR -> NOGR .RELS**	0	4	completor (16) in (4)
(21)	**SATZ -> NOGR .VERB**	0	4	completor (17) in (2)
(22)	**NOGR -> NOGR .RELS**	0	4	completor (17) in (4)
(23)	**RELS -> .RELW VERB**	4	4	predictor (18) und (20) und (22)

Section 5:

(24)	**SATZ -> NOGR VERB.**	0	5	scanner for (19), *fehlen*
(25)	**SATZ -> NOGR VERB.**	0	5	scanner for (21), *fehlen*
(26)	**# -> .SATZ**	0	5	completor (24) in (1)
(27)	**# -> .SATZ**	0	5	completor (25) in (1)

Warum gibt es zwei Ergebnisse? Einmal wird erst das ADJE und dann der RELS mit der NOGR *Ideen* verknüpft, das andere Mal erst der RELS und dann das ADJE. Linguistisch ist der Unterschied nicht gerechtfertigt.





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

2. Grammatik der deutschen Kardinalzahlen in Chomsky-Normalform. Fettgedruckte Kategorien markieren jeweils die dominierende Konstituente (den *Head*) in der Regel. Tiefgestellte Zahlen sind Subklassifizierungen. Kommata trennen alternative Werte voneinander ab.

Lexikon:	**Regeln:**	**Bereich der Regeln:**
Z_1 = {ein, zwei, drei, vier, fünf, sechs, sieben, acht, neun}	(R-1) Z_2 -> Z_3 + **zehn**	11 - 19
Z_2 = {zehn, elf, zwölf}	(R-2) Z_7 -> Z_4 + **zig**	20 - 90, außer 30
Z_3 = {drei, vier, fünf, sech, sieb, acht, neun}	(R-3) Z_7 -> drei + **ssig**	30
Z_4 = {zwan, vier, fünf, sech, sieb, acht, neun}	(R-4) Z_8 -> U + $\mathbf{Z_7}$	21 - 99
Z_5 = {hundert}	(R-5) Z_9 -> Z_1 + $\mathbf{Z_5}$	100 - 900
Z_6 = {tausend}	(R-6) Z_{10} -> Z_2 + $\mathbf{Z_5}$	1100 - 1900 (elfhundert)
drei = {drei}	(R-7) Z_{11} -> $\mathbf{Z_{5,9}}$ + $Z_{1,2,7,8}$	100 - 199, 200 - 299 usw. bis 999
zehn = {zehn}	(R-8) Z_{12} -> $\mathbf{Z_{10}}$ + $Z_{1,2,7,8}$	1101 - 1999 (elfhundertein)
zig = {zig}	(R-9) Z_{13} -> $Z_{1,2,7,8,9,11}$ + $\mathbf{Z_6}$	1000 - 999000
ssig = {ssig}	(R-10) Z_{14} -> $\mathbf{Z_{6,13}}$ + $Z_{1,2,7,8,9,11}$	1999 - 999999
und = {und}	(R-11) U -> Z_1 + **und**	1 - 9 und ... zig

2. Protokoll der Analyse

zwei	*hundert*	*zwei*	*und*	*zwan*	*zig*	*tausend*	*vier*	*hundert*	*sieb*	*zehn*
Z1	Z5	Z1	und	Z4	zig	Z6	Z1,3,4	Z5	Z3,4	Z2, zehn
0 1	1 2	2 3	3 4	4 5	5 6	6 7	7 8	8 9	9 10	10 11

WORKING TABLE

Section 1:

	Category	**Left margin**	**Right margin**	**Left IC**	**Right IC**	**Explanation**
(1)	**Z1**	0	1	-	-	shift *zwei*

Section 2:

(2)	**Z5**	1	2	-	-	shift *hundert*
(3)	**Z9**	0	2	1	**2**	reduce by R-5

Section 3:

(4)	**Z1**	2	3	-	-	shift *zwei*
(5)	**Z11**	1	3	**2**	4	reduce by R-7
(6)	**Z11**	0	3	**3**	4	reduce by R-7

Section 4:

(7)	**und**	3	4	-	-	shift *und*
(8)	**U**	2	4	4	7	reduce by R-11

Section 5:

(9)	**Z4**	4	5	-	-	shift *zwan*

Section 6:

(10)	**zig**	5	6	**-**	-	shift *zig*
(11)	**Z7**	4	6	**9**	10	reduce by R-2
(12)	**Z8**	2	6	**8**	11	reduce by R-4
(13)	**Z11**	1	6	**2**	12	reduce by R-7
(14)	**Z11**	0	6	**3**	12	reduce by R-7

Section 7:

(15)	**Z6**	6	7	**-**	-	shift *tausend*
(16)	**Z13**	4	7	11	**15**	reduce by R-9
(17)	**Z13**	2	7	12	**15**	reduce by R-9
(18)	**Z13**	1	7	13	**15**	reduce by R-9
(19)	**Z13**	0	7	14	**15**	reduce by R-9

Section 8:

(20)	**Z1**	7	8	**-**	-	shift *vier*
(21)	**Z3**	7	8	**-**	-	shift *vier*
(22)	**Z4**	7	8	**-**	-	shift *vier*
(23)	**Z14**	6	8	**15**	20	reduce by R-10
(24)	**Z14**	4	8	**16**	20	reduce by R-10
(25)	**Z14**	2	8	**17**	20	reduce by R-10
(26)	**Z14**	1	8	**18**	20	reduce by R-10
(27)	**Z14**	0	8	**19**	20	reduce by R-10

Section 9:

(28)	**Z5**	8	9	-	-	shift *hundert*
(29)	**Z9**	7	9	20	**28**	reduce by R-5
(30)	**Z14**	6	9	**15**	29	reduce by R-10
(31)	**Z14**	4	9	**16**	29	reduce by R-10
(32)	**Z14**	2	9	**17**	29	reduce by R-10
(33)	**Z14**	1	9	**18**	29	reduce by R-10
(34)	**Z14**	0	9	**19**	29	reduce by R-10

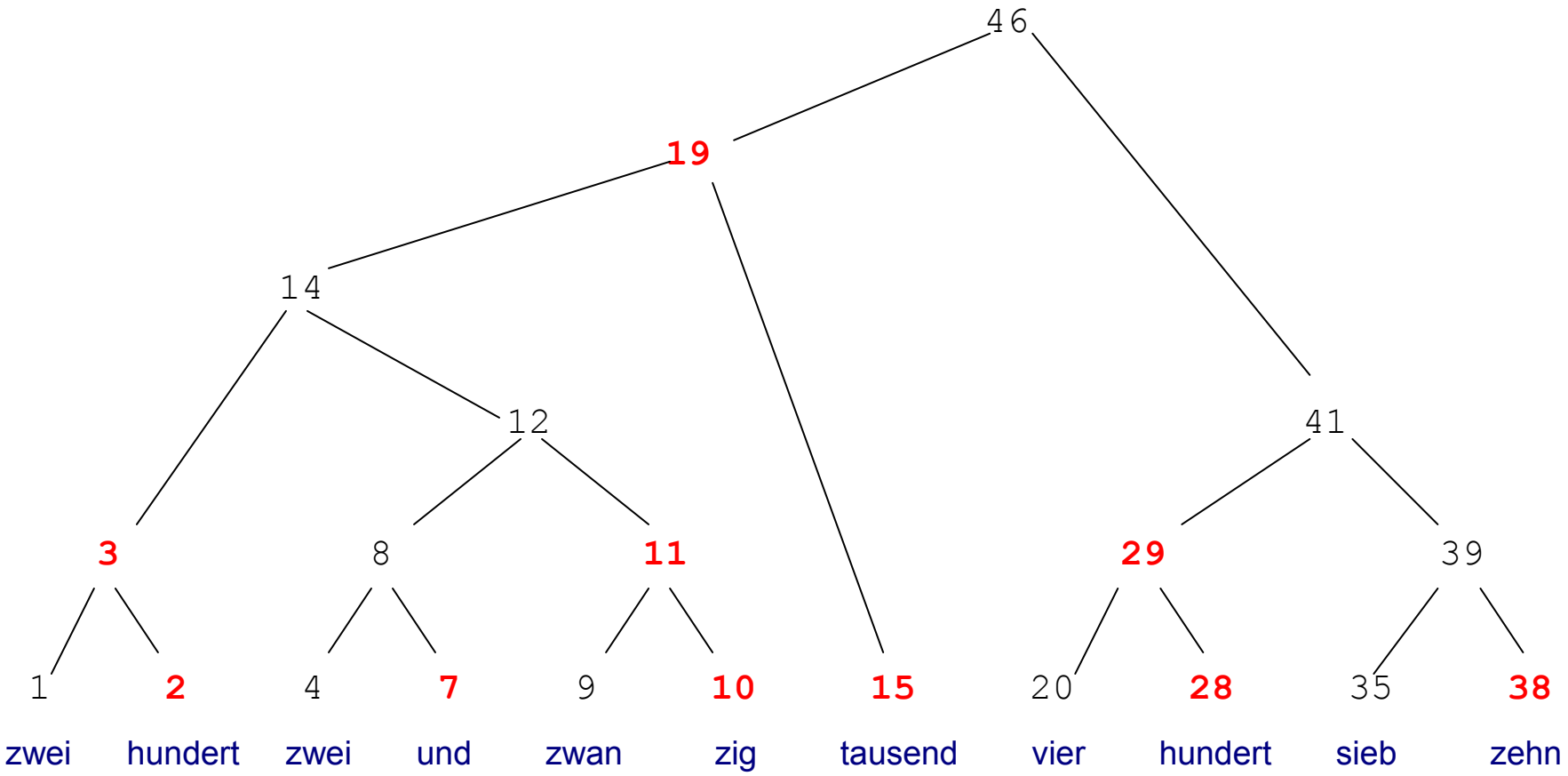
Section 10:

(35)	**Z3**	9	10	-	-	shift *sieb*
(36)	**Z4**	9	10	-	-	shift *sieb*

Section 11:

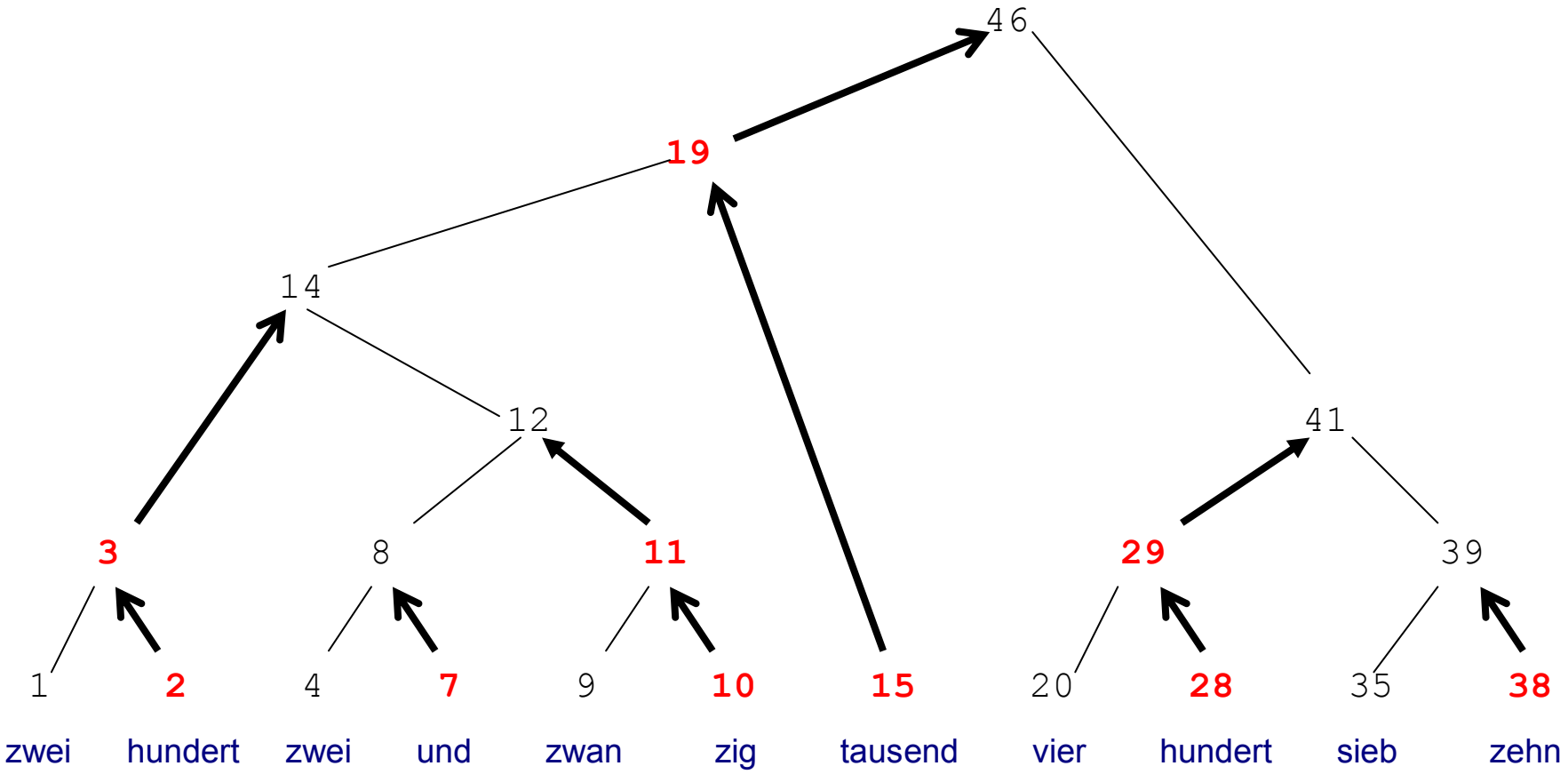
(37)	**Z2**	10	11	-	-	shift *zehn*
(38)	**zehn**	10	11	-	-	shift *zehn*
(39)	**Z2**	9	11	35	**38**	reduce by R-1
(40)	**Z11**	8	11	**28**	39	reduce by R-7
(41)	**Z11**	7	11	**29**	39	reduce by R-7
(42)	**Z14**	6	11	**15**	41	reduce by R-10
(43)	**Z14**	4	11	**16**	41	reduce by R-10
(44)	**Z14**	2	11	**17**	41	reduce by R-10
(45)	**Z14**	1	11	**18**	41	reduce by R-10
(46)	**Z14**	0	11	**19**	41	reduce by R-10

3. Konstituenzbaum nach der Working Table (Heads rot und fett)

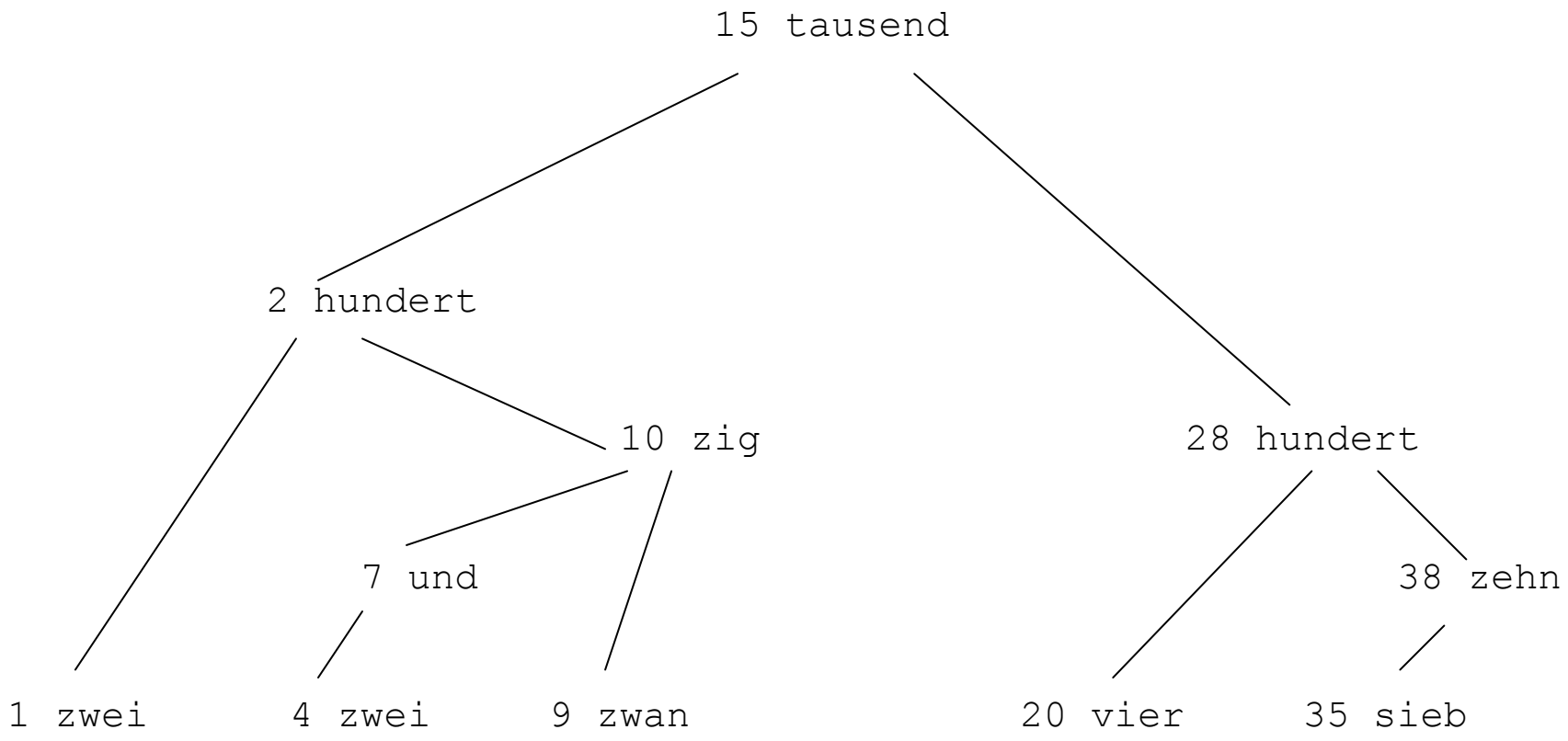


Konstruktion eines Dependenzbaums

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:


15 tausend
2 hundert
10 zig
28 hundert
7 und
38 zehn
1 zwei
4 zwei
9 zwan
20 vier
35 sieb

Lösung 8: Chart-Parsen mit einer Kategorialgrammatik (PT-5)

Die Grammatik aus Aufgabe 6:

	Regeln:		**Lexikon:**	
(R-1)	SATZ -> NOGR VERB		Ideen	NOGR
(R-2)	NOGR -> ADJE NOGR		begeistern	VERB
(R-3)	NOGR -> NOGR RELS		fehlen	VERB
(R-4)	RELS -> RELW VERB		farblose	ADJE
			neue	ADJE
			die	RELW

Umformung in eine Kategorialgrammatik:

Alte lexikalische Kategorie:	**Neue Lexikalische Kategorien:**	**Erklärung:**
VERB	NOGR\SATZ	nach (R-1)
	RELW\RELS	nach (R-4)
NOGR	NOGR	atomar
	NOGR/RELS	nach (R-3)
ADJE	NOGR/NOGR	nach (R-2)
RELW	RELW	atomar

Ergebnis, Kategorialgrammatik, die nur noch aus Lexikoneinträgen besteht:

Ideen	NOGR, NOGR/RELS
begeistern	NOGR\SATZ, RELW\RELS
fehlen	NOGR\SATZ, RELW\RELS
farblose	NOGR/NOGR
neue	NOGR/NOGR
die	RELW

Eingabe:	*neue*	*Ideen*	*die*	*begeistern*	*fehlen*
Lexikon:	NOGR/NOGR	NOGR, NOGR/RELS	RELW	NOGR\SATZ, RELW\RELS	NOGR\SATZ, RELW\RELS
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)	**NOGR/NOGR**	0	1	-	-	scanner *neue*

Section 2:

(2)	**NOGR, NOGR/RELS**	1	2	-	-	scanner *Ideen*

Section 3:

(4)	**RELW**	2	3	-	-	scanner *die*

Section 4:

(6)	**NOGR\SATZ, RELW\RELS**	3	4	-	-	scanner *begeistern*
(7)	**RELS**	2	4	4	6	completed 6 by 4
(8)	**NOGR**	1	4	2	7	completed 2 by 7
(9)	**NOGR**	0	4	1	8	completed 1 by 8

Section 5:

(11)	**NOGR\SATZ, RELW\RELS**	4	5	-	-	scanner *fehlen*
(12)	**SATZ**	1	5	8	11	completed 11 by 8
(13)	**SATZ**	0	5	9	11	completed 11 by 9

Vergleich mit PT-4: Der Bottom-up Chart Parser (PT-5) kombiniert mit einer Kategorialgrammatik erscheint erheblich effizienter als der Top-Down Chart Parser (13 gegenüber 27 Schritten).

Zur Bottom-up Strategie passen eben besonders gut lexikalisierte Grammatiken.

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 → . NP VP NP → . Noun NP → . Det Noun NP → . Adj Noun NP → . Pron	
(z-1)	S' -> S.		acc
(z-2)	S → NP . VP	VP → . Vt NP VP → . Vtger GP	
(z-3)	S → NP VP .		re 1
(z-4)	NP → Noun .		re 2
(z-5)	NP → Det . Noun		
(z-6)	NP → Det Noun .		re 3
(z-7)	NP → Adj . Noun		
(z-8)	NP → Adj Noun .		re 4

(z-9)	NP → Pron .		re 5
(z-10)	VP → Vt . NP	NP → . Noun NP → . Det Noun NP → . Adj Noun NP → . Pron	
(z-11)	VP → Vt NP .		re 6
(z-12)	VP → Vtger . GP	GP → . Ger NP	
(z-13)	VP → Vtger GP .		re 7
(z-14)	GP → Ger . NP	NP → . Noun NP → . Det Noun NP → . Adj Noun NP → . Pron	
(z-15)	GP → Ger NP .		re 8

Funktion FOLLOW

```
FOLLOW(S)  = $
FOLLOW(NP) = VP, FOLLOW(VP), FOLLOW(GP)
FOLLOW(VP) = FOLLOW(S) = $
FOLLOW(GP) = FOLLOW(VP) = FOLLOW(S) = $
```

Funktion FIRST

```
FIRST(NP) =  Noun, Det, Adj, Pron
FIRST(VP)  = Vt, Vtger
FIRST(GP)  = Ger
FIRST(S)  = FIRST(NP) = Noun, Det, Adj, Pron
FIRST($)  =  $
```

ACTION TABLE

z	Det	Adj	Noun	Pron	Vt	Vtger	Ger	$
0	sh5	sh7	sh4	sh9				
1								acc
2					sh10	sh12		
3								re1
4					re2	re2		re2
5			sh6					
6					re3	re3		re3
7			sh7					
8					re4	re4		re4
9					re5	re5		re5
10	sh5	sh7	sh4	sh9				
11								re6
12							sh14	
13								re7
14	sh5	sh7	sh4	sh9				
15								re8

GOTO TABLE

NP	VP	GP	S
2			1
	3		
	11		
		13	
15			

2. Test

Input:	students	hate	annoying	their	professors
terminale Kategorien:	**Noun**	**Vt/Vtger**	**Adj/Ger**	**Det**	**Noun**
Position:	1	2	3	4	5

P:	States and Descriptions:	Explanation:
1	**0**	starting state
2	**0 -Noun- 4**	sh4
2	**0 -NP(Noun)- 2**	re2
3	**0 -NP(Noun)- 2 -|** **|-Vt- 10** **|-Vtger- 12**	sh10 sh12
4	**0 -NP(Noun)- 2 -|** **|-Vt- 10 -Adj- 7** **|-Vtger- 12 -Ger- 14**	sh7 (fails) sh14
5	**0 -NP(Noun)- 2 -Vtger- 12 -Ger- 14 -Det- 5**	sh5

```
6    0 -NP(Noun)- 2   -Vtger   12   -Ger- 14 -Det- 5 -Noun- 6    sh5

6    0 -NP(Noun)- 2   -Vtger   12   -Ger- 14 NP(Det Noun) - 15   re3

6    0 -NP(Noun)- 2   -Vtger   12   GP(Ger (NP(Det Noun)) - 13   re8

6    0 -NP(Noun)- 2   VP(Vtger GP(Ger (NP(Det Noun))) - 3       re7

6    0 S(NP(Noun) VP(Vtger GP(Ger (NP(Det Noun)))) - 1          re7

acc
```

3. Evaluierung:

Durch die Vorausschau ist der PT-6 erheblich effizienter als der PT-1.

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

1. Umformung des Netzes in ein deterministisches Übergangsnetzwerk:

Ausgangszustand:	Eingabesymbol:	Zielzustand:
1/2/6a	A1	2/6
1/2/6a	V	3/4/7/8
2/6	V	3/4/7/8
3/4/7/8e	E1	4
3/4/7/8e	#	5/6
3/4/7/8e	E2	8/e
4	#	5/6
5/6	A2	6/2
5/6	V	3/4/7/8
6/2	V	3/4/7/8

Einsetzung der elementaren Symbole für die Mengen:

Z1	Eingabesymbol	Z2
1/2/6a	b | bl | br | c | ch | chl | chr | cl | cr | d | dr | dsch | f | fl | fr | g | gh | gl | gn | gr | h | j | k | kl | kn | kr | l | m | n | p | pf | pfl | pfr | ph | phr | pl | pr | ps | qu | r | rh | s | sch | schl | schm | schn | schr | schw | sk | skl | skr | sl | sm | sn | sp | sph | spl | spr | st | str | sz | t | th | thr | tr | tsch | v | w | wr | x | z | zw	2/6
1/2/6a	a | ä | aa | ae | ah | äh | ai | äo | au | äu | ay | e | ea | ee | eh | ei | eo | eu | ey | i | ia | ie | ieh | ih | io | o | ö | oa | oh | öh | oi | oo | öo | ou | ou | oua | oui | oy | u | ü | ua | uh | üh | ui | y	3/4/7/8
2/6	a | ä | aa | ae | ah | äh | ai | äo | au | äu | ay | e | ea | ee | eh | ei | eo | eu | ey | i | ia | ie | ieh | ih | io | o | ö | oa | oh | öh | oi | oo | öo | ou | ou | oua | oui | oy | u | ü | ua | uh | üh | ui | y	3/4/7/8
3/4/7/8e	ch | d | f | g | k | l | m | n | p | ph | r | s | sch | ß | st | t | th | v | x | z	4
3/4/7/8e	#	5/6
3/4/7/8e	b | bs | bsch | bst | bt | ch | chs | chst | cht | chts | chz | chzt | ck | c-k | cks | ckst | ckt | d | ds | dst | dt | f | ff | ffs | ffst | fft | fs | fst | ft | fts | fzt | g | gd | gg | ggst | ggt | gs | gst | gt | k | ks | kst | kt | kts | l | lb | lbs | lbst | lbt | lch | ld | lds | lf | lfs | lfst | lft | lg | lgs | lgst | lgt | lk | lks | lkst | lkt | ll | lls | llst | llt | lm | lms | ln | lnd | lp | ls | lsch | lscht | lst | lt | lts | ltst | lz | lzt | m | mb | mbst | mbt | md | mf | mm | mms | mmst | mmt | mp | mpf | mpfst | mpft | mps | mpst | ms | msch | mschst | mscht | mst | mt | mts | n | nch | nd | nds | ndt | nf | nft | nfts | ng | ngs | ngst | ngt | nk | nks | nkst | nkt | nkts | nn | nns | nnst | nnt | ns | nsch | nst | nt | nts | nz | nzt | p | pf | pfs | pfst | pft | ph | pp | pps | ppst | ppt | ps | pt | pts | r | rb | rbs | rbst | rbt | rch | rchst | rcht | rd | rds | rf | rfs | rfst | rft | rg | rgs | rgst | rgt | rk | rks | rkst | rkt | rl | rls | rlst | rlt | rm | rmst | rmt | rn | rnd | rns | rnst | rnt | rp | rps | rpt | rr | rrschst | rrscht | rrst | rrt | rs | rsch | rst | rt | rts | rts | rv | rvst | rvt | rz | rzt | s | sch | schst | scht | sk | ß | st | t | th | ts | tsch | tt | tts | ttst | ttst | tz | tzt | v | vs | x | xt | z | zt | zz	8/e
4	#	5/6
5/6	b | c | ch | d | f | g | h | j | k | l | m | n | p | qu | r | s | sch | ß | st | t | v | w | x | z	6/2
5/6	a | ä | aa | ae | ah | äh | ai | äo | au | äu | ay | e | ea | ee | eh | ei | eih | eo | eu | ey | i | ia | ie | ieh | ih | io | o | ö | oa | oh | öh | oi | oo | öo | ou | ou | oua | oui | oy | u | ü | ua | uh | üh | ui | y	3/4/7/8
6/2	a | ä | aa | ae | ah | äh | ai | äo | au | äu | ay | e | ea | ee | eh | ei | eo | eu | ey | i | ia | ie | ieh | ih | io | o | ö | oa | oh | öh | oi | oo | öo | ou | ou | oua | oui | oy | u | ü | ua | uh | üh | ui | y	3/4/7/8
8e	=	1/2/6a

2. Abarbeitung der Eingaben:

wal#ten#de

Ausgangszustand:	Eingabesymbol:	Zielzustand:
1/2/6a	w	2/6
2/6	a	3/4/7/8e
3/4/7/8e	l	4
4	#	5/6
5/6	t	6/2
6/2	e	3/4/7/8
3/4/7/8	n	4
4	#	5/6
5/6	d	6/2
6/2	e	3/4/7/8

wald=en#de

Ausgangszustand:	Eingabesymbol:	Zielzustand:
1/2/6a	w	2/6
2/6	a	3/4/7/8e
3/4/7/8e	ld	8e
8e	=	1/2/6a
1/2/6a	e	3/4/7/8
3/4/7/8	n	4
4	#	5/6
5/6	d	6/2
6/2	e	3/4/7/8

wal#te#nde

Ausgangszustand:	Eingabesymbol:	Zielzustand:
1/2/6a	w	2/6
2/6	a	3/4/7/8e
3/4/7/8e	l	4
4	#	5/6
5/6	t	6/2
6/2	e	3/4/7/8
3/4/7/8	#	5/6
5/6	n	6/2
6/2	d	kein Übergang, fail

3. Reicht der endliche Erkenner zur Lösung der Aufgabe, Wörter in Silben zu parsen?

   Leider nein. Da sich die Zeichenketten in A1, A2, E1 und E2 überlappen, ist die Grammatik in Wirklichkeit nicht deterministisch. Man benötigt ein Übergangsnetzwerk mit Backtracking, d.h. ein RTN statt eines FTN.

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

1. ATN-Programm:

```
node A
arc 1: CAT zahl
     test <GETF typ> equals "2"
     action SETR hunderter <GETF zif>
     transition TO B
arc 2: WRD hundert
     action SETR hunderter "1"
     transition TO C
arc 3: JUMP
     transition TO C

node B
arc 4: WRD hundert
     transition TO C

node C
arc 5: CAT zahl
     test <GETF typ> equals "1"
```

```
    test <GETF zif> equal "10" then
    action SETR zehner "1"
    else test <GETF zif> equal "11" then
    action SETR zehner "1"
    action SETR einer "1"
    else test <GETF zif> equal "12" then
    action SETR zehner "1"
    action SETR einer "2"
    else action SETR einer <GETF zif>
    transition TO H

arc 6: CAT zahl
    test <GETF typ> equals "3"
    action SETR einer <GETF zif>
    transition TO D

arc 7: CAT zahl
    test <GETF typ> equals "2"
    action SETR einer <GETF zif>
    transition TO E

arc 8: JUMP
    transition TO F

arc 9: JUMP
    transition TO H
```

node D

arc 10: WRD zehn

```
    action SETR zehner "1"
    transition TO H
```

node E

arc 11: WRD und

```
    transition TO F
```

node F

arc 12: CAT zahl

```
    test <GETF type> equals "4"
    action SETR zehner <GETF zif>
    transition TO G
```

node G

arc 14: WRD zig

```
    test <GETF zif> not equal "3"
    transition TO H
```

arc 15: WRD ssig

```
    test <GETF zif> equals "3"
    transition TO H
```

node H

arc 16: POP ziffer

```
test <GETR hunderter> not empty and
<GETR zehner> empty then
action SETR zehner "0"
else T
test <GETR zehner> not empty and
<GETR einer> empty then
action SETR einer "0"
else T
BUILDQ ziffer + + + <hunderter> <zehner> <einer>
```

2. Trace für die Zahl vierhundertfünfunddreissig

P:	Word:	Configurations:
1	**vier**	*node* **A** *arc 1:* CAT zahl *test* <GETF typ> equals "2" = T SETR hunderter 4 *transition* TO B
2	**hundert**	*node* **B** *arc 4:* WRD hundert *transition* TO C
3	**fünf**	*node* **C** *arc 5:* CAT zahl *test* <GETF typ> equals "1" = T test <GETF zif> equal "10" = F test <GETF zif> equal "11" = F test <GETF zif> equal "12" = F SETR einer 5 transition TO H
4	**und**	*node* **H** Backtracking
3	**fünf**	*node* **C** *arc 6:* CAT zahl test <GETF typ> equals "3" = T SETR einer 5 transition TO D

```
4    und     node D
                Backtracking
3    fünf    node C
                transition TO NextCharacter
                arc 7: CAT zahl
                test <GETF typ> equals "2" = T
                SETR einer 5
                transition TO E

4    und     node E
                arc 11: WRD und
                transition to F

5    drei    node F
                arc 12: CAT zahl
                arc 2: WRD V fails
                test <GETF type> equals "4" = T
                SETR zehner 3
                transition TO G

6    ssig    node G
                arc 14,15: WRD ssig
                test <GETF zif> equals "3" = T
                transition TO H

             node H
                arc 16: POP ziffer
                test <GETR hunderter> not empty and <GETR zehner> empty =F
                test <GETR zehner> not empty and <GETR einer> empty = F
                T
                BUILDQ ziffer 4 3 5 <hunderter> <zehner> <einer>
```

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

Eingabe: *vier-hundert-fünf-und-drei-ssig*

1. Analyse:

A:	Segment:	Left margin:	Right margin:	Head bulletin:	Filler bulletin:
(1)	**vier**	1	1	-	-
(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_zig] wert[4])					
(2)	**hundert**	2	2	-	-
(kategorie[zahl] typ[hundert] wert[100] (< SLOT rolle[MULTP] kategorie[zahl] typ[ein, zehn]) (> SLOT rolle[ADDIT] kategorie[zahl] typ[ein, zehn, zig, ssig]))					
(3)	**vierhundert**	1	2	2	1
(kategorie[zahl] typ[hundert] wert[100] (< rolle[MULTP] kategorie[zahl] typ[ein] wert[4]) (> SLOT rolle[ADDIT] kategorie[zahl] typ[ein, zehn, zig, ssig]))					
(4)	**fünf**	3	3	-	-
(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_zig] wert[5])					

(5)	**vierhundertfünf**	1	3	3	4	

(kategorie[zahl] typ[hundert] wert[100]
(< rolle[MULTP] kategorie[zahl] typ[ein] wert[4])
(> rolle[ADDIT] kategorie[zahl] typ[ein] wert[5]))

(6)	**und**	4	4	–	–	

(kategorie[zahl] typ[und]
(< SLOT kategorie[zahl] typ[vor_und]))

(7)	**fünfund**	3	4	4	6	

(kategorie[zahl] typ[und]
(< kategorie[zahl] typ[vor_und] wert[5]))

(8)	**drei**	5	5	–	–	

(kategorie[zahl] typ[ein, vor_und, vor_zehn, vor_ssig] wert[3])

(9)	**ssig**	6	6	–	–	Y

(kategorie[zahl] typ[zig] wert[10]
(< SLOT rolle[MULTP] kategorie[zahl] typ[vor_ssig])
(< SLOT rolle[ADDIT] kategorie[zahl] typ[und]))

(10)	**dreissig**	5	6	9	8

(kategorie[zahl] typ[zig] wert[10]
(< rolle[MULTP] kategorie[zahl] typ[vor_ssig] wert[3])
(< SLOT rolle[ADDIT] kategorie[zahl] typ[und]))

(11)	**fünfunddreissig**	3	6	10	7

(kategorie[zahl] typ[zig] wert[10]
(< rolle[MULTP] kategorie[zahl] typ[vor_ssig] wert[3]))
(< rolle[ADDIT]kategorie[zahl] typ[und]
(< kategorie[zahl] typ[vor_und] wert[5])))

(12)	**vierhundert fünfunddreissig**	1	6	3	11

(kategorie[zahl] typ[hundert] wert[100]
(< rolle[MULTP] kategorie[zahl] typ[ein] wert[4])
(> rolle[ADDIT] kategorie[zahl] typ[ssig] wert[10]
(< rolle[MULTP] kategorie[zahl] typ[vor_ssig] wert[3])
(< rolle[ADDIT]kategorie[zahl] typ[und]
(< kategorie[zahl] typ[vor_und] wert[5]))))

2. Die Zahl 435 erhält man, wenn man im Dependenzbaum zunächst die Werte der MULTP-Elemente mit ihren Heads multipliziert und anschließend die Werte der ADDIT-Elemente zu ihren Hedas addiert.